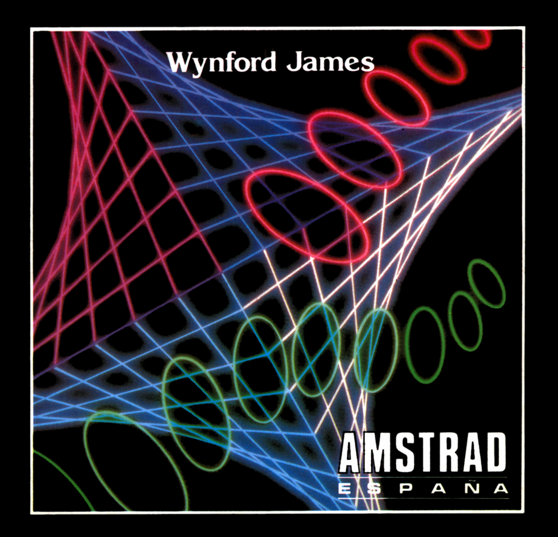
PROGRAMACION BASIC CON AMSTRAD



PROGRAMACIÓN BASIC CON AMSTRAD

PROGRAMACIÓN BASIC CON AMSTRAD

Wynford James



PROGRAMACIÓN BASIC CON AMSTRAD

edición española de la obra

BASIC PROGRAMMING ON THE AMSTRAD
Wynford James

publicada en castellano bajo licencia de

MICRO PRESS
Castle House
27 London Road
Tunbridge Wells, Kent

Traducción

Emilio Benito Santos

INDESCOMP, S.A.
Avda. del Mediterráneo, 9
28007 Madrid

ISBN 84 86176 **25** 5
Depósito legal: M-10945-85

Impresión:

Gráficas Lormo. Isabel Méndez, 15. Madrid.

Producción de la edición española:

Vector Ediciones.
Gutierre de Cetina, 61.
28017 Madrid (91) 408 52 17

Contenido

1

Primeros pasos

INTRODUCCIÓN AL SISTEMA DEL MICROORDENADOR

Ya es usted propietario de un microordenador Amstrad y, como tal, se encuentra en una situación privilegiada, porque lo que usted ha adquirido es un completo sistema de microordenador. Es decir, no va a necesitar comprar nada más para poder servirse de él desde el momento en que lo desembale.

¿Qué es un sistema de microordenador? Su Amstrad se aloja en una caja gris de plástico que contiene numerosos componentes electrónicos que le permiten tanto dibujar como realizar operaciones aritméticas, o llevar a cabo todas las demás tareas que hoy día exigimos de los microordenadores personales.

Pero el ordenador por sí solo, sin su ayuda, es una criatura básicamente ignorante; tan torpe como un coche sin conductor. Para que el Amstrad pueda realizar aun la más mínima tarea útil, ha de recibir instrucciones de un ser humano. Para ello el Amstrad dispone de un teclado que nos permite comunicarnos con el microprocesador situado en su interior y pasarle así nuestras instrucciones.

Sin embargo, los seres humanos tenemos puntos débiles que el ordenador no tiene. Por ejemplo, a nosotros nos gusta ver lo que estamos escribiendo; por eso el Amstrad dispone de un monitor, especie de pantalla de televisión en la que aparece cada letra que vamos pulsando en el teclado.

El medio que nos permite dar instrucciones al Amstrad es el teclado, o dispositivo de entrada de datos. El ordenador también puede comunicarnos mensajes, haciéndolos aparecer en el monitor, o dispositivo de salida. De ordenar los datos se ocupa el propio ordenador. Entonces ¿no necesitamos nada más? ¿Para qué sirve el cassette que se encuentra a la derecha del teclado?

Para poder entender por qué es tan útil el cassette que forma parte esencial del Amstrad, tenemos que examinar más de cerca los componentes del ordenador. Aunque estos componentes son muy numerosos, para nosotros los más importantes son los que componen su memoria.

EL ORDENADOR OLVIDADIZO

El Amstrad tiene dos tipos de memoria: la ROM y la RAM. Quizá sea más fácil comprender por qué necesitamos ambas si examinamos las cosas que recuerdan los seres humanos: nuestros nombres, dónde vivimos, cómo conectar el televisor. Todo

esto está permanentemente almacenado en nuestro cerebro, ya que nos es imprescindible para nuestra vida cotidiana. El ordenador también tiene que recordar permanentemente ciertas instrucciones: tiene que saber aritmética, qué puntos de la pantalla tiene que iluminar cuando se pulsa en el teclado la letra 'A', cómo dibujar rectas, y muchas otras cosas. El fabricante del Amstrad ha grabado las instrucciones que capacitan al ordenador para realizar estas tareas en ROM (*read-only memory*, memoria de sólo lectura).

Se dice que esta memoria es de "sólo lectura" porque el acceso que a ella tienen tanto el ordenador como el usuario está limitado al mero examen o lectura de la ROM. Estas instrucciones están grabadas permanentemente y no se las puede cambiar. Esto es deseable, ya que no nos interesa interferir accidentalmente con la capacidad de cálculo que posee el ordenador, ni que, por ejemplo, aparezca en la pantalla una 'Q' cada vez que tecleemos la 'A'.

Por otra parte, los seres humanos no recuerdan permanentemente todo lo que hacen. A veces se alegran de poder olvidar aquella pésima película que vieron anoche en la televisión o las malas notas que obtuvieron en un examen, o la distancia de la Tierra a la Luna. Gran parte de la información que se utiliza se recuerda solamente durante un tiempo muy corto; por ejemplo, el número de teléfono de la tienda de ordenadores, el nombre de las personas que han batido cierto record de atletismo, etc.

Del mismo modo, el ordenador recuerda cierta información sólo temporalmente. Las instrucciones registradas en ROM hacen posible que el ordenador multiplique 12345 por 6789, pero, una vez realizada la operación y mostrado el resultado en el monitor, ¿tendría alguna utilidad que almacenase para siempre la respuesta?

Verdaderamente no, y por consiguiente el ordenador almacena las instrucciones que nosotros le damos en RAM (*random-access memory*, memoria de acceso aleatorio). Ésta es una memoria temporal. La RAM está vacía cuando conectamos la máquina, y se va ocupando paulatinamente según vamos introduciendo nuestras instrucciones. Esta información permanecerá en el ordenador hasta que lo apaguemos.

La RAM realiza la función de un encerado en el que el ordenador registra la información, la cual se borra cuando lo desconectamos. La RAM es capaz de contener una cantidad de información que equivaldría a unas 2Ø páginas de este libro. Naturalmente, el microordenador no nos sería muy útil si sólo pudiese almacenar información de modo permanente, ya que pronto agotaríamos toda su memoria y su efectividad. De ahí que podamos borrar la RAM en cualquier momento, pidiendo al ordenador que lo haga o, simplemente, desconectándolo.

Por otra parte, la facilidad con que podemos borrar la RAM nos plantea otro problema. Supongamos que hemos estado trabajando con el ordenador varias horas. Hemos escrito muchas instrucciones, denominadas *programa*, que nos permiten dibujar una vista de la Tierra divisada desde el espacio. ¿Qué hacemos ahora con esta obra maestra? Si dejamos el ordenador conectado permanentemente para conservar el programa, no podremos volver a utilizarlo, ya que otro programa necesitará la RAM que hemos ocupado. El dilema es que si desconectamos el ordenador perdemos el programa almacenado en la RAM.

Así es como llegamos, por un camino indirecto, a lo que explica la presencia del cassette situado a la derecha del teclado. El ordenador puede convertir cualquier

programa almacenado en RAM en una serie de señales eléctricas que podemos grabar en cinta magnetofónica. Una vez grabado el programa, podemos desconectar el Amstrad con toda tranquilidad.

La próxima vez que queramos demostrar a nuestros amigos que somos unos grandes artistas, todo lo que tenemos que hacer es pasar por la máquina la cinta que contiene el programa; el ordenador se ocupará de convertir los sonidos grabados en programa para la RAM, esto es, de "cargar" el programa. Podemos pedir al ordenador que obedezca las instrucciones del programa, es decir, que lo ejecute, y la pantalla del monitor mostrará de nuevo la vista de la Tierra desde el espacio.

Cabe mencionar aquí que los diferentes modelos de microordenadores no pueden cargar los programas de otros modelos.

EL AMSTRAD

Ahora que nos hemos familiarizado un poco con los principales componentes de un sistema informático y hemos visto cuál es su utilidad, examinemos el Amstrad.

El teclado del Amstrad es similar al de una máquina de escribir, si bien está dotado de algunas teclas adicionales, de diferente color que el resto del teclado. Vamos a estudiar para qué sirven esas teclas. Conecte su Amstrad (las instrucciones de instalación aparecen en las páginas F1.1 a F1.6 de la Guía del Usuario que se suministra con el ordenador).

Si no ha utilizado nunca un microordenador, quizá le preocupe la posibilidad de averiar una máquina tan costosa tecleando algo indebidamente. No tema: nada que usted haga en el teclado puede afectar permanentemente al Amstrad. Si la pantalla presenta un color rojo subido y no muestra nada de que lo usted escribe, lo peor que puede ocurrir es que tenga que desconectar el ordenador. Cuando vuelva a conectarlo, el problema habrá desaparecido.

Al encender su Amstrad, verá en la pantalla una líneas de texto que le anuncian que está utilizando un ordenador Amstrad, y a continuación, en otra línea, 'BASIC 1.Ø'. BASIC es el lenguaje de ordenador utilizado por la mayor parte de los microordenadores, y en concreto por el Amstrad. Algo más abajo en la pantalla aparecerá el mensaje 'Ready' (preparado), y en la siguiente línea, un pequeño rectángulo, junto al borde izquierdo de la pantalla. Este rectángulo se denomina *cursor de texto*; indica el lugar en que va a aparecer el siguiente carácter que se escriba; equivale, pues, al punto de la pluma.

Cuando usted conecta el Amstrad, las teclas están en situación de minúsculas, aunque las letras que están grabadas en las teclas sean mayúsculas. Así, cuando pulse las teclas A, B, C observará cómo el Amstrad escribe en la pantalla:

```
abc■
```

Observe también que el cursor se desplaza a medida que usted va escribiendo y que siempre indica la *próxima* posición en que el ordenador va a escribir un carácter.

Pulse ahora la tecla verde CAPS LOCK de la izquierda del teclado. Pulse a continuación las teclas A, B y C de nuevo y podrá ver en la pantalla:

```
abcABC■
```

La tecla CAPS LOCK cambia todas las letras del alfabeto a mayúsculas. Si la pulsamos otra vez, volvemos al modo de minúsculas. Así, pulsando CAPS LOCK por segunda vez y tecleando A, B y C tendremos:

```
abcABCabc■
```

Como se puede observar, CAPS LOCK funciona como conmutador que bascula entre los modos de mayúsculas y minúsculas. Sin embargo, no hay forma de saber en qué situación se encuentra el conmutador con sólo mirar al teclado. El único modo de averiguarlo es pulsar una tecla y observar el resultado.

Si cometemos errores

En realidad no ocurre nada grave si se pulsa una tecla equivocadamente o si se escribe en mayúsculas cuando se deseaba hacerlo en minúsculas; estos errores se corrigen muy fácilmente en el Amstrad. Pulse la tecla verde de la parte superior derecha del teclado, marcada con DEL, y no la suelte.

El cursor regresa al comienzo de la línea, borrando todos los caracteres que encuentra a su paso. La tecla DEL es la tecla de borrado (*delete* en inglés), que hace desaparecer el carácter que se encuentre a la izquierda del cursor. Si se mantiene esta tecla pulsada durante algún tiempo, su acción se repite automáticamente y continúa borrando caracteres hasta que se la suelta o hasta que el cursor llega al principio de la línea.

Análogamente ocurre con casi todas las teclas del Amstrad: se autorrepiten tras una breve pausa. Pulse la tecla A y no la suelte mientras en la pantalla sigan apareciendo letras 'a'. Observe que se han completado seis líneas y que el cursor se ha detenido en la séptima.

La razón por la que el ordenador detiene la repetición automática de la tecla es porque ya tenemos 255 caracteres y el Amstrad no permite líneas de mayor longitud que ésta. Para el Amstrad, el borde derecho de la pantalla no es el final del renglón, de modo que, aunque lo que hemos escrito ocupe más de seis líneas físicas de la pantalla, para el ordenador es una sola línea continua.

Para el Amstrad, la línea sólo termina cuando se lo indicamos explícitamente. Podemos decirle "éste es el final de esta línea" pulsando la tecla grande azul que está marcada con ENTER. Hágalo y vea qué sucede.

Después de la línea de letras 'a' aparece el siguiente mensaje:

```
Syntax error
Ready
■
```

Cuando pulsamos la tecla ENTER, le estamos pidiendo al ordenador que dé por terminada la línea que estábamos escribiendo y que la introduzca en la memoria. En

este momento estamos usando el Amstrad en modo *inmediato* o *directo*, así denominado porque cada vez que pulsamos ENTER el ordenador examina la línea que acabamos de introducir y obedece inmediatamente las instrucciones contenidas en ella. En todo caso, el ordenador ha de entender las instrucciones para poder obedecerlas. En concreto, no ha encontrado ningún sentido a la línea de 255 letras 'a'; las palabras 'Syntax error' son su forma de decir "no entiendo". La palabra 'Ready' sólo significa que el ordenador ha terminado de obedecer todas las órdenes y está preparado para recibir más.

El objetivo de este libro es estudiar las instrucciones que el Amstrad sí entiende; pero antes de pasar a explicarlas, vamos a completar el breve examen del teclado.

Las dos teclas verdes marcadas con SHIFT sirven para seleccionar los caracteres de la parte superior de las teclas que tienen dos símbolos. Teclee los números 1234567890 (situados en la fila superior del teclado) y después, manteniendo pulsada una de las dos teclas SHIFT, teclee 1234567890 de nuevo:

```
1234567890!"#$%&'()_■
```

Cualquier tecla que esté marcada con dos caracteres hará que aparezca en la pantalla el de la mitad superior, si la pulsamos al mismo tiempo que una de las teclas SHIFT. Aunque el conmutador CAPS LOCK se encuentre en situación de minúsculas, al pulsar SHIFT al mismo tiempo que una letra alfabética, ésta aparecerá en mayúscula.

Las letras marcadas con dos símbolos producen normalmente el de la mitad inferior. Hay otro conmutador que cambia esta situación: funciona pulsando CAPS LOCK al tiempo que se mantiene pulsada la tecla verde CTRL situada en el extremo inferior derecho del teclado. Hágalo y pulse a continuación las teclas de la primera fila del teclado:

```
!"#$%&'()_■
```

Las teclas alfabéticas escribirán ahora en mayúsculas. No obstante, puede continuar escribiendo números también, ya que el Amstrad tiene un teclado independiente, situado a la derecha del principal, destinado a este fin. Para normalizar el teclado se debe pulsar nuevamente la combinación CTRL/CAPS LOCK.

La tecla DEL es útil cuando se observa un error poco después de cometerlo, esto es, cuando el carácter incorrecto está cerca del cursor. Pero ¿qué ocurre si el error está al principio de la línea y no queremos perderla? Escriba lo siguiente *sin* pulsar la tecla ENTER:

```
Ezte es un error
```

Tenemos que poner una 's' en lugar de la 'z' de 'Ezte'. Recordemos que el cursor nos indica en qué posición aparecerá el siguiente carácter. La posición del cursor puede ser modificada utilizando las teclas marcadas con flechas y situadas por encima del teclado numérico. Por eso se las denomina *teclas del cursor*. Pulse la tecla ← y no la suelte. El cursor se desplazará hacia la izquierda de la línea. Suéltela cuando el cursor esté sobre la letra 'z' de 'Ezte'. No se preocupe si se pasa; con

la tecla → podrá volver a llevar el cursor al punto deseado. La 'z' será visible a través del cursor. Si pulsamos ahora la tecla CLR, situada junto a la DEL en la parte superior derecha del teclado, la 'z' desaparecerá y tendremos:

```
Ete es un error
```

Los restantes caracteres de la línea han retrocedido un lugar y el cursor se encuentra ahora sobre la 't' de 'Ete'. Para borrar cualquier otro carácter de la línea seguiríamos el mismo procedimiento: situar el cursor sobre el carácter incorrecto y pulsar CLR. Si tenemos varios caracteres equivocados seguidos, manteniedo pulsada la tecla CLR haremos que ésta se autorrepita y los borre todos. DEL borra el carácter situado a la izquierda del cursor, mientras que CLR hace desaparecer el que está situado bajo el cursor.

En este ejemplo queremos también intercalar un nuevo carácter en la palabra 'Ete'. Desplacemos el cursor hasta la 't' y pulsemos la tecla S. Así insertamos una 's' en el lugar correcto y el resto de la línea se desplaza un lugar hacia la derecha:

```
Este es un error
```

Los errores que se cometen al escribir la líneas son fáciles de corregir usando las teclas CLR, DEL y las del cursor, pero es preciso recordar que este sistema funciona siempre que no se haya pulsado antes ENTER. Una vez pulsada esta tecla, el ordenador considera que la línea está terminada. Para corregir (o, como se dice en la jerga informática, *editar*) líneas que ya hayan sido introducidas en la memoria, hay que seguir un procedimiento diferente.

Las teclas CTRL, ESC y COPY realizan funciones especiales que se comprenderán mejor cuando hayamos practicado algo más con el Amstrad. La tecla TAB no tiene ningún cometido especial; sólo produce una flecha que apunta a la derecha.

Cómo dar órdenes al Amstrad

Ya hemos visto que para el Amstrad una línea de 255 letras 'a' no tiene ningún sentido. ¿Qué tipo de instrucciones puede entender y obedecer este ordenador? Son muchas y se las daremos a conocer más adelante. Hay órdenes que hacen que el Amstrad dibuje rectas en la pantalla, cambie los colores, realice cálculos complicados o emita notas musicales. Todas estas órdenes tienen algo en común: el ordenador las obedece porque el fabricante ha grabado los pormenores de estas órdenes en la memoria ROM, y por lo tanto el ordenador puede reconocerlas.

Pero esto da lugar a que el ordenador nos pueda parecer tanto muy listo como muy torpe, según los casos. Puede realizar maravillas y, no obstante, por una sola letra incorrecta, se negará a obedecer y hará que en la pantalla aparezca el mensaje 'Syntax error' para indicarnos que está perdido. Vamos a ver una de las órdenes que el Amstrad puede obedecer: la instrucción PRINT (imprimir, escribir). Un humano quizás aceptaría PRONT en lugar de PRINT (después de todo, la 'i' y la 'o' están juntas en el teclado y es fácil cometer este error). Pero el ordenador no aceptaría PRONT.

Cuando demos órdenes al Amstrad tendremos que atenernos a las reglas de *sintaxis* del lenguaje de ordenador que él entiende. Ustedpuedeentenderestafraseaun-cuandonotengaespacios, pero no el Amstrad.

La visualización en la pantalla

Ya estamos preparados para comenzar a utilizar el Amstrad seriamente. Lo primero que tenemos que hacer es reinicializar (*reset*) el ordenador; es decir, devolverlo al estado en que se encontraba en el momento de encenderlo, con la RAM completamente borrada. Para ello podemos desconectarlo y volverlo a conectar; también podemos obtener el mismo resultado, de modo menos drástico, pulsando la tecla ESC en combinación con CTRL y SHIFT. La pantalla se borra y vuelve a aparecer el anuncio de Amstrad.

Cuando estamos seguros de que la línea que hemos escrito, es decir, nuestra "entrada" (*input*) hacia el ordenador, es correcta, pulsamos la tecla ENTER para indicar al Amstrad que hemos acabado y que ya puede obedecer la orden. Por el momento escribiremos ⟨ENTER⟩ para recordarle que debe pulsar esta tecla cada vez que termine de escribir una instrucción. Escriba ahora lo siguiente:

```
mode2<ENTER>
```

y obtendrá el mensaje:

```
Syntax error
Ready
■
```

El Amstrad no ha entendido la orden. En el lenguaje BASIC del Amstrad se utilizan los espacios para indicar dónde termina una palabra y dónde comienza la siguiente. El Amstrad puede comprender 'mode' si le sigue un espacio, pero en este caso cree que la palabra es 'mode2', que él no conoce. Los espacios son *muy importantes* en el BASIC de Amstrad, por lo que deberá asegurarse de que no omite ninguno en los siguientes ejemplos. En algunos casos descubrirá usted por sí mismo que los espacios no parecen tener importancia en algunas instrucciones (por ejemplo, precediendo a las dobles comillas). De todos modos, lo más seguro es espaciar siempre las palabras ya que, cuando los espacios no son necesarios, el Amstrad sencillamente los ignora, mientras que si lo son y usted los omite, el Amstrad se quejará. Escriba ahora:

```
mode 2<ENTER>
```

En esta ocasión el ordenador reconoce la palabra 'mode' y obedece la instrucción. La pantalla se borra y aparece el mensaje "Ready", en letras mucho más estrechas, en el extremo superior izquierdo de la pantalla.

En el mundo real hay diversas clases de papel destinadas a usos diferentes. El

arquitecto no diseña una casa en un cuaderno de notas, y el novelista no utiliza papel de tamaño A3 para escribir sus relatos. En informática, la pantalla equivale a la hoja de papel, y por lo tanto es útil poder elegir el formato que más nos convenga. La instrucción 'mode' seguida de los números Ø, 1 o 2 selecciona uno de los formatos de que dispone el Amstrad. Cada modo permite la visualización de un número diferente de caracteres por línea.

Si el texto que tenemos que escribir es muy extenso, nos será útil poder visualizar el mayor número de caracteres posible. El modo 2 es el más adecuado en este caso, ya que en este modo el Amstrad escribe en una pantalla de 25 líneas por 8Ø columnas.

Teclee ahora:

```
mode 1<ENTER>
```

La pantalla se borrará de nuevo y aparecerá el mensaje 'Ready' en el extremo superior izquierdo de la pantalla. Este modo ya nos es familiar, pues es el que el Amstrad adopta cuando lo encendemos o reinicializamos. El modo 1 tiene 25 líneas de 4Ø caracteres cada una. En este modo los caracteres son más legibles; podemos considerarlo como modo "de trabajo" para la introducción de instrucciones.

Escriba ahora lo siguiente:

```
mode Ø<ENTER>
```

De nuevo aparecerá 'Ready', pero ahora en caracteres mucho más anchos. El modo Ø nos da 25 líneas de 2Ø caracteres cada una. Este modo es el más adecuado para dibujar en color, como veremos más adelante.

Modo	Número de líneas	Caracteres por línea
Ø	25	2Ø
1	25	4Ø
2	25	8Ø

Figura. 1. Los tres modos de pantalla del Amstrad.

Casi todos los ordenadores son bastante exigentes en cuanto a la utilización de mayúsculas y minúsculas cuando escribimos las instrucciones, pero una de las buenas características del Amstrad es que acepta igualmente 'mode Ø', 'MODE Ø' o incluso 'mOde Ø' como órdenes válidas. Por razones que veremos más adelante, es aconsejable utilizar siempre letras minúsculas. En el resto del libro aparecerán instrucciones en mayúsculas y en minúsculas para demostrar que son intercambiables, pero le recomendamos que se acostumbre a escribirlas en minúsculas. Finalmente, escriba

```
mode 1<ENTER>
```

para hacer que el ordenador vuelva al "modo de trabajo".

La instrucción PRINT

Escriba:

```
print"Pedro"<ENTER>
```

En la pantalla podrá ver:

```
Pedro
Ready
█
```

Utilizamos PRINT cuando queremos visualizar algo en la pantalla. Esta instrucción escribe, sin modificación alguna, cuanto pongamos entre comillas. Teclee

```
print"7+5"<ENTER>
```

y obtendrá

```
7+5
Ready
█
```

El ordenador no ha calculado el valor de 7+5 porque la expresión estaba entre comillas, y todo lo que encuentra entre comillas lo copia literalmente.

Usemos ahora el Amstrad como si fuera una calculadora, sirviéndonos de PRINT para visualizar los resultados. Escriba:

```
print 7+5<ENTER>
```

(El signo '+' se obtiene de la tecla del ';' combinada con SHIFT.) El ordenador escribirá:

```
 12
```

Esta vez ha calculado la suma y escrito el resultado, porque los números no estaban entre comillas. El espacio que precede al 12 está reservado para el signo; los números mayores que Ø aparecen con un espacio delante, en lugar del signo +. Teclee:

```
print 7-5<ENTER>
```

(El signo – se encuentra en la misma tecla que el =.) El ordenador escribirá:

```
2
```

Ahora pruebe

```
print 5-7<ENTER>
```

que dará

```
-2
```

La multiplicación es un poco más peculiar. En lugar del aspa, ×, los ordenadores suelen utilizar el asterisco, *, como signo de multiplicar. Escriba:

```
print 9*7<ENTER>
```

(El signo * se obtiene de la tecla ':' combinada con SHIFT.) El ordenador escribirá:

```
63
```

El signo de división es la barra inclinada, /, que está en la tecla del signo de interrogación, '?'. Escriba:

```
print 12/4<ENTER>
```

y obtendrá

```
3
```

Hasta ahora sólo hemos hecho aritmética muy sencilla, pero naturalmente el ordenador puede realizar cálculos mucho más complicados. Escriba:

```
print 12345*9876<ENTER>
```

y obtendrá

```
121919220
```

O, con tres factores,

```
print 12.34*5.67*8.9<ENTER>
```

lo cual dará

```
622.71342
```

Después de realizar el cálculo anterior, observará que el contenido de la pantalla se ha desplazado hacia arriba. La instrucción 'print "Pedro" ' con que empezamos esta sección ha desaparecido. El ordenador desplaza la pantalla hacia arriba cuando el cursor ha alcanzado la última línea y todavía tiene que seguir escribiendo.

En este momento la pantalla está bastante llena. Vamos a borrarla, aprovechando al mismo tiempo para presentar otra instrucción de BASIC. Escriba:

```
cls<ENTER>
```

Estas letras son la abreviatura de *CLear Screen* (borrar la pantalla); su efecto no requiere más explicación. A partir de ahora, utilice 'cls<ENTER>' siempre que quiera borrar la pantalla.

Orden y claridad en la pantalla

Dado que tendrá que utilizar la instrucción PRINT tan frecuentemente para programar el Amstrad, le alegrará saber que la ROM reconoce el signo '?' como abreviatura de 'print'. Escriba:

```
?"El signo ? es abreviatura de print"<ENTER>
```

(El signo ? se obtiene con / y SHIFT.) Escriba:

```
?"Si suma 7 y 6 obtiene";"el resultado"<ENTER>
```

y en la pantalla aparecerá:

```
Si suma 7 y 6 obtieneel resultado
```

El signo ';' le pide al Amstrad que siga escribiendo y además que lo haga en la misma línea. Pero, al no haber dejado espacio entre 'observe' y las comillas en la instrucción, se nos han juntado las dos palabras: 'obtieneel'. No ha sido demasiado útil, ¿verdad? Pruebe entonces con esto:

```
?"Si suma 7 y 6 obtiene";7+6<ENTER>
```

En la pantalla ha aparecido:

```
Si suma 7 y 6 obtiene 13
```

Así está mejor. El ordenador ha realizado el cálculo y mostrado el resultado. Pero escriba:

```
?"Si calcula 5-6 obtiene";5-6;"lo cual e
s confuso"<ENTER>
```

En la pantalla aparece:

```
Si calcula 5-6 obtiene-1
lo cual es confuso
```

Todo esto demuestra el cuidado que hay que tener con el ';' en las instrucciones PRINT. Pruebe ahora con los espacios en su sitio:

```
?"Si suma 7 y 6 obtiene ";"el resultado"<ENTER>
?"Si calcula 5-6 obtiene ";5-6;"lo cual
no es confuso"<ENTER>
```

Escriba:

```
?1;2;3<ENTER>
```

y observe que en

```
1  2  3
```

el ordenador ha puesto un espacio antes de cada número, reservado para un posible signo –, y otro después. Pruebe ahora con:

```
?12;13;14<ENTER>
```

lo que da:

```
12  13  14
```

Como puede observar, estos números no quedan alineados en vertical con los que escribimos antes. Si estuviésemos tratando de escribir números en columnas, esta forma de visualizarlos sería confusa. Probemos con comas en lugar de con el signo de punto y coma; escriba:

```
?1,2,3<ENTER>

 1             2             3
```

y ahora:

```
?12,13,14<ENTER>

 12            13            14
```

Esto tampoco es perfecto porque, por ejemplo, podríamos no querer tanta separación entre columnas, pero al menos los números han quedado bien alineados. La coma ha pedido al ordenador que saltase a la siguiente "zona de escritura" de la pantalla. El mismo método vale para palabras:

```
?"Pedro","Juan","Elena"<ENTER>
Pedro          Juan           Elena
```

Si cree que los nombres no están bien alineados con los números, recuerde que éstos van precedidos de un espacio.

Organización de la pantalla en modo 1

¿Cómo sabe el Amstrad en qué lugar de la pantalla debe escribir para mantener la alineación en vertical? Dicho de otro modo, ¿qué son las "zonas de escritura"? Ya sabemos que en el modo 1 la pantalla consta de 25 líneas de 4Ø caracteres cada una.

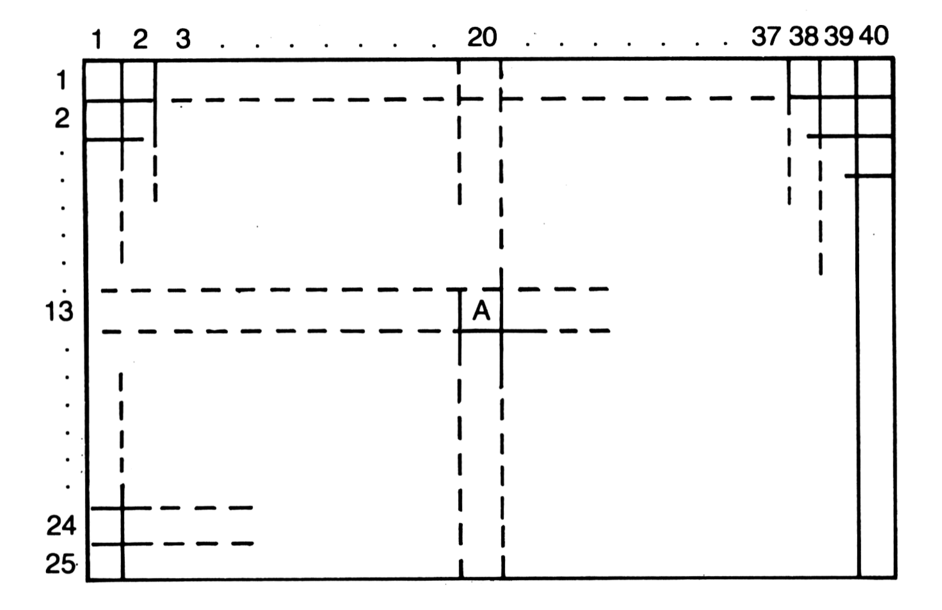


Figura 2. La pantalla en modo 1.

Podemos escribir un carácter en cualquier lugar de la pantalla siempre que identifiquemos su posición mediante dos números: el número de columna y el de fila. La letra 'A' de la figura 2 se encuentra en la columna 2Ø y en la línea 13, aproximadamente en el centro de la pantalla. Observe que siempre mencionamos el número de columna antes que el de fila.

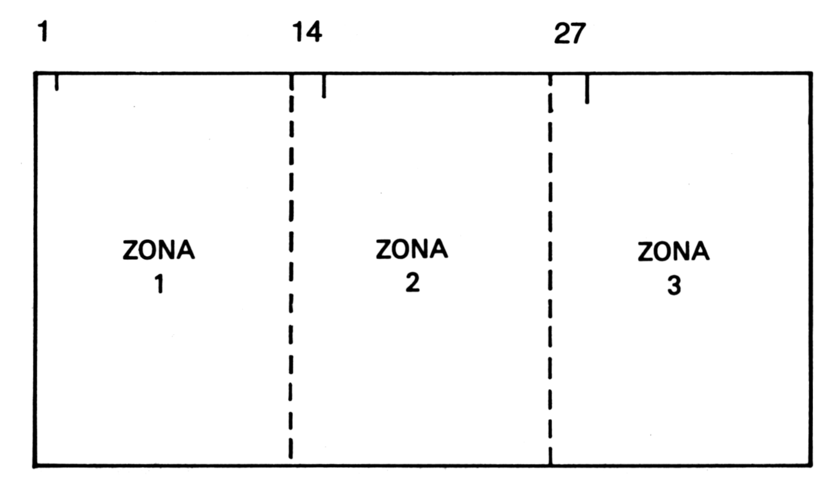


Figura 3. Las zonas de escritura en modo 1.

El Amstrad divide la pantalla en *zonas de escritura* verticales, cada una de ellas de 13 caracteres de anchura. Si en una instrucción PRINT separamos los elementos con comas, el primero se empieza a escribir en la columna 1, el segundo en la 14 y el tercero en la 27. Para el cuarto tendría que empezar en la columna 40 pero, como ya no hay más sitio para otra zona de 13 caracteres, saltará a la columna 1 de la línea siguiente. Si alguno de los elementos es de más de 13 caracteres, abarcando, por lo tanto, dos o más zonas, el siguiente elemento se escribe en la siguiente zona libre. Escriba:

```
?1,2,3,4,5<ENTER>
```

y verá cómo el ordenador escribe los dos últimos números en una nueva línea. Pruebe con

```
?"Demasiado largo.","Cabe.","Pero este n
o cabe."<ENTER>
```

para ver cómo se las arregla el ordenador cuando tiene que escribir algo que no cabe en una zona de escritura.

Las instrucciones TAB y SPC

Aunque las zonas de escritura son útiles para escribir los datos en forma de tabla en la pantalla, sería una limitación grave estar restringidos a ellas. El BASIC de Amstrad dispone de intrucciones que nos permiten elegir a nuestro gusto las posiciones de escritura en la pantalla. Así, para escribir un título centrado podemos utilizar la instrucción PRINT TAB o, abreviadamente, ?TAB. Escriba lo siguiente:

```
cls<ENTER>
?tab(6)"Esto empieza en la columna 6"<ENTER>
```

Si consultamos la figura 2, veremos que en modo 1 disponemos de 4Ø columnas. Podemos hacer referencia a ellas por sus números.

Con la instrucción ?TAB(6) estamos pidiendo al ordenador que empiece a escribir a partir de la columna número 6. En modo 1, el número que va entre paréntesis después de 'TAB' puede ser cualquiera comprendido entre 1 y 4Ø.

Quizá quiera usted experimentar con números mayores que 4Ø, o con números negativos.

Dentro de una misma instrucción PRINT se pueden incluir varias cláusulas TAB:

```
?tab(6)"Buenos"tab(18)"dias"tab(3Ø)"amig
os"<ENTER>
```

La instrucción TAB ordena avanzar hasta la posición especificada escribiendo espacios; así, cuando el ordenador ha escrito "Buenos", escribe espacios hasta alcanzar la columna 18, en la cual empieza a escribir "dias". Se podría pensar que la instrucción

```
?tab(3Ø)"amigos"tab(18)"dias"tab(6)"Buen
os"<ENTER>
```

debería producir el mismo efecto que la anterior, pero no es así. La cláusula TAB no hace retroceder la posición de escritura dentro de una línea, sino que previamente provoca el avance a la línea siguiente.

Escriba ahora:

```
?"La"spc(4)"frase"spc(4)"queda"spc(4)"mu
y"spc(4)"espaciada"<ENTER>
```

El ordenador deja 4 espacios entre cada dos palabras. Con SPC(1Ø) dejaría 1Ø espacios, y así sucesivamente. El parámetro de SPC, es decir, el número que va entre paréntesis, puede estar entre 1 y 4Ø en modo 1. SPC y TAB se pueden mezclar en una misma instrucción PRINT:

```
?tab(8)"Una mezcla de TAB"spc(8)"y SPC"<ENTER>
```

Como habrá observado, las cláusulas TAB y SPC impiden el salto a la línea siguiente lo mismo que si se tratara de signos de punto y coma ';'.

Ejercicios

1. Escriba varias instrucciones PRINT, utilizando como separador la coma, en los modos Ø y 2. ¿Por qué son estos resultados diferentes de los que se obtienen en modo 1?
2. Experimente con TAB y SPC en modo 2 y en modo Ø. ¿Hay diferencias entre los márgenes de valores que podemos utilizar en los diversos modos?

2

La programación

PROGRAMACIÓN SENCILLA

En el capítulo anterior hemos estado utilizando el ordenador en modo *inmediato* o *directo*. Cada vez que escribíamos una instrucción, el Amstrad la obedecía inmediatamente, si la entendía. El modo directo es útil porque nos permite experimentar con las órdenes y observar los resultados inmediatamente.

Pero lo más normal es utilizar el ordenador en modo *de programa*. Para observar la diferencia entre ambos modos, escriba lo siguiente:

```
1 print"Manolo"<ENTER>
```

Al pulsar la tecla ENTER no ha ocurrido nada evidente. Como hemos escrito un número, el 1, al principio de la línea, el ordenador lo ha considerado como *número de línea* y ha supuesto, por consiguiente, que no tenía que ejecutar la instrucción inmediatamente, sino almacenarla en la RAM en espera de que le pidamos que la ejecute.

Para comprobar que efectivamente la instrucción ha quedado almacenada en la RAM, escriba:

```
list<ENTER>
```

En la pantalla ha aparecido el listado del programa, que de momento consiste en solamente una línea. El ordenador reconoce la palabra 'print' como una de las *palabras clave* de BASIC y por lo tanto la ha convertido en PRINT, en mayúsculas. Escriba ahora:

```
9 print"Enrique"<ENTER>
5 print"Ricardo"<ENTER>
list<ENTER>
```

A pesar de que hemos introducido las líneas en el orden 1, 9, 5, en el listado aparecen en orden de números crecientes, y en ese mismo orden las ejecutará cuando llegue el momento. ¿Cómo podemos hacer para que el ordenador ejecute el programa? Escriba:

```
run<ENTER>
```

El ordenador ejecuta el programa, esto es, obedece las instrucciones del mismo por orden creciente de número de línea. Cuando ha terminado de ejecutarlo, el programa permanece almacenado en la memoria, como podrá comprobar escribiendo de nuevo LIST<ENTER>. Teclee:

```
5 print"Elena"<ENTER>
list<ENTER>
```

Observará que la nueva línea ha sustituido a la antigua, y que el pobre Ricardo ha desaparecido. Cuando desee efectuar un cambio en cualquier línea, escriba simplemente la nueva línea con el mismo número de la que desee sustituir. Teclee:

```
1<ENTER>
list<ENTER>
```

La línea ha sido reemplazada por una línea vacía, por lo que el Amstrad no la incluye ya como parte del programa. De este fácil modo podemos eliminar cualquier línea que no necesitemos.

Ahora podemos hacer varias cosas con el programa: ampliarlo añadiendo nuevas líneas en cualquier orden, conservarlo para uso futuro grabándolo en la cinta o borrarlo de la RAM. Esto último es lo que vamos a hacer, hasta que tengamos un programa que merezca la pena. Teclee:

```
new<ENTER>
```

Esto indica al ordenador que se desea borrar el programa y comenzar otro nuevo. Si tecleamos ahora list<ENTER> no aparecerá ningún listado, ya que el programa se ha borrado.

AYUDAS QUE PUEDE OFRECERLE EL AMSTRAD

El Amstrad posee diversas características que facilitan considerablemente la programación.

Lo prudente es espaciar los números de las líneas, en previsión de que más tarde haya que intercalar alguna instrucción entre las ya introducidas. Se complicarán las cosas si usted ha dado a sus líneas los números 1, 2, 3 y 4 y luego se le ocurre intercalar una nueva línea entre la 2 y la 3. Por eso es habitual numerar las líneas de diez en diez; de esta forma dejamos números libres que más tarde podemos asignar a las líneas adicionales. La numeración automática de las líneas es una labor que el Amstrad puede realizar por usted. Utilizaremos este método para escribir el siguiente programa. Teclee:

```
auto<ENTER>
```

El ordenador escribirá "1Ø" y esperará a que usted introduzca la línea. Teclee:

```
10 mode 0<ENTER>
20 ?"Esto esta en modo 0"<ENTER>
30 ?"Aqui tiene una multiplicacion:"<ENTER>
40 ?"73.45 por 5.769 da";73.45*5.769<ENTER>
```

El ordenador escribirá tambien el número 5Ø después de la última línea, pero ya no tenemos más líneas que añadir al programa. Para detener la numeración automática de las líneas, debemos pulsar la tecla roja ESC ("escape"), situada en la parte superior izquierda del teclado. Teclee ahora:

```
list<ENTER>
```

y observará que la línea 5Ø no está en el programa. Teclee:

```
run<ENTER>
```

y verá cómo el ordenador obedece su programa línea por línea. Teclee:

```
mode 1<ENTER>
```

para volver al modo "de trabajo".

AUTO puede empezar la numeración a partir de cualquier número de línea. Además, podemos elegir también el intervalo entre líneas, aunque generalmente se espacian del modo arriba indicado. Teclee:

```
auto 15,1<ENTER>
```

y observará que los números de las líneas comenzarán por el 15 y se incrementarán de uno en uno. Teclee:

```
15?"He aqui una linea mas"<ENTER>
16?"y aqui otra"<ENTER>
```

y pulse ⟨ESC⟩ de nuevo. Liste el programa y ejecútelo. Verá que las dos nuevas líneas han sido insertadas en el programa en la posición correcta. El ordenador comprueba si los números de línea que va generando han aparecido ya en el programa, ya que en tal caso las líneas nuevas sustituyen a las antiguas de igual número en cuanto pulsamos ⟨ENTER⟩. El ordenador le advierte de este riesgo mostrando[1] un asterisco * después del número de línea. Compruébelo tecleando:

```
auto<ENTER>
```

[1] En el modelo CPC664 aparece la línea completa, la cual puede ser editada o conservada sin modificación.

El ordenador escribirá:

```
1Ø*
```

para advertirle de que ya tiene una línea con el número 1Ø y de que, si escribe algo antes de pulsar ⟨ENTER⟩, la línea quedará sustituida por lo que escriba. Si no desea modificar el programa, pulse ⟨ESC⟩.

Si sigue intercalando líneas en el programa, podrá llegar a encontrarse con el problema que intentaba evitar: que no quede lugar para nuevos números de línea. Si lista el presente programa, verá que ya no hay hueco para otra línea entre la 15 y la 16. Afortunadamente, el Amstrad puede ayudarnos también en este caso. Teclee:

```
renum<ENTER>
list<ENTER>
```

La orden RENUM vuelve a numerar todas las líneas del programa, empezando por la número 1Ø e incrementando los números de línea de diez en diez. Al igual que con AUTO, se puede elegir tanto el número de la primera línea como el intervalo entre líneas. Teclee:

```
renum 1ØØ,5<ENTER>
list<ENTER>
```

y verá cómo el programa queda renumerado comenzando por la línea 1ØØ y con los siguientes números de línea espaciados de 5 en 5.

¿Qué podemos hacer si cometemos un error?

Escriba *exactamente* lo siguiente:

```
1 pront"Esta es la primera linea"<ENTER>
```

En esta línea hemos cometido un error deliberadamente, al escribir 'pront' en lugar de 'print'. En cuanto pulsamos ⟨ENTER⟩ en modo *inmediato*, el Amstrad intenta obedecer la orden; si la orden es errónea, el ordenador emite un *mensaje de error* para informar de lo que ha ocurrido.

Esto no ocurre así en modo *de programa*. Lo que el Amstrad hace en este caso es almacenar cualquier línea que esté numerada y no comprueba si es inteligible mientras no le pidamos que ejecute el programa. Escriba:

```
run<ENTER>
```

El ordenador tratará de obedecer las instrucciones de la línea 1, pero se ve obligado a abandonar la ejecución del programa porque no reconoce 'pront'. Le dirá cuál es la línea que está causando el problema, e incluso la muestra en la pantalla. Usted

puede ahora *editar* la línea 1 para corregir el error.

Como puede observar, el cursor se encuentra sobre el primer carácter de la línea, o sea, sobre el número 1. Utilizando la tecla de *cursor a la derecha* (la tecla con flecha situada junto a la tecla verde de COPY de la parte superior derecha del teclado), desplace el cursor hasta que se encuentre sobre la 'o' de 'pront'. En el capítulo anterior vimos que pulsando CLR borrábamos el carácter situado en la posición del cursor. Esto es lo que tenemos que hacer ahora, ya que debemos borrar la 'o'. Pulse CLR una vez. La línea queda de la siguiente forma:

```
1 prnt"Esta es la primera linea"
```

con el cursor sobre la 'n'. Ahora tenemos que insertar la letra 'i'. El Amstrad inserta automáticamente cualquier carácter que tecleemos, *delante* (a la izquierda) del carácter que se encuentra bajo el cursor. Pulse ahora 'i' y podrá leer:

```
1 print"Esta es la primera linea"
```

Los cambios que hemos realizado hasta ahora han afectado sólo al aspecto de la línea en pantalla. Pulse ⟨ENTER⟩ y el Amstrad sustituirá la línea incorrecta por esta nueva versión corregida. (En este caso no importa que el cursor no se encuentre al final de la línea.) Si quiere comprobar que la línea ha quedado corregida, liste otra vez el programa.

Si observa que ha cometido un error en su programa, antes de ejecutarlo, puede corregir la línea fácilmente tecleando:

```
edit 1<ENTER>
```

o el número de cualquier otra línea que desee editar. Después puede utilizar las teclas del cursor, la CLR o quizás la DEL para corregir la línea. Para practicar esta técnica, introduzca algunas modificaciones en las líneas de su programa y devuelva después las líneas a su forma original.

La edición mediante el copiado de líneas

Hay otro método de edición, denominado "del cursor de copia", que se basa en copiar líneas de la pantalla. Escriba lo siguiente:

```
1 pront"He aqui otra linea con error"<ENTER>
```

Pulse la tecla de 'cursor hacia arriba' (↑) al tiempo que pulsa una de las teclas SHIFT. Ha aparecido un segundo cursor, que se ha superpuesto al 1 de la línea que acaba de escribir. Éste es el cursor de copia, que señala lo que vamos a copiar. El otro es el cursor normal de texto, que muestra en qué lugar de la pantalla va a aparecer lo que copiemos. Pulse la tecla COPY una vez y verá que el '1' queda copiado en la nueva línea. Pulse COPY tres veces más y verá lo siguiente en su pantalla:

```
1 pront"He aqui otra linea con error"
1 pr
```

Al llegar aquí no deseamos copiar la 'o', pues ésta es la parte incorrecta de la línea. Como el cursor inferior es el de texto, o sea, el que normalmente muestra dónde se va a escribir el siguiente carácter que pulsemos, si pulsamos ahora la 'i' ésta aparecerá en esta posición y la pantalla mostrará:

```
1 pront"He aqui otra linea con error"
1 pri
```

Ahora tenemos que copiar el resto de la línea, pero sin la 'o'. Si piensa que puede desplazar el cursor de copia utilizando sin más las teclas del cursor, pruebe y verá lo que ocurre.

Las teclas del cursor mueven el cursor de texto, pero el Amstrad emitirá un pitido si usted intenta salirse de la línea nueva que está confeccionando. El cursor de copia se mueve siempre manteniendo pulsada una tecla SHIFT al tiempo que pulsa las de movimiento del cursor. Hágalo así para colocar el cursor de copia sobre la 'n' de 'pront' saltándose la 'o'. Asegúrese de que el cursor de texto esté ***a la derecha*** de la 'i' de '1 pri', ya que es ahí donde el ordenador comenzará a escribir el texto copiado. Mantenga pulsada la tecla COPY hasta que llegue al final de la línea que está copiando. No se preocupe si copia también espacios, porque el Amstrad no los tendrá en cuenta. Pulse ⟨ENTER⟩ para que la nueva línea 1 sustituya a la antigua en la memoria del ordenador, lo que podrá comprobar listando el programa.

En este momento este proceso puede parecerle complicado, pero se familiarizará con él rápidamente. Merece la pena que dedique algún tiempo a aprender a editar, ya que así podrá programar con rapidez mucho mayor.

Las mil y una aplicaciones de la tecla COPY

Aunque la tecla COPY es muy útil para editar, también se la puede aprovechar para introducir líneas nuevas. Si en el programa hay varias líneas similares, se puede copiar trozos de una línea a otra. Veamos un ejemplo sencillo. Escriba:

```
new<ENTER>
auto<ENTER>
10 print"*******"<ENTER>
```

Ahora utilice las teclas del cursor combinadas con la de SHIFT para poner el cursor sobre la "p" de "print". Pulse la tecla COPY y copie el resto de la línea; al final, pulse ⟨ENTER⟩. Ahora debe de haber en la pantalla lo siguiente:

```
10 print"*******"<ENTER>
20 print"*******"<ENTER>
30
```

Repita este procedimiento de copia para las líneas 3Ø, 4Ø y 5Ø. Pulse ESC en cuanto aparezca el número de la línea 6Ø. Liste y ejecute el programa. El resultado deberá ser un rectángulo formado por asteriscos. Podemos utilizar esta misma técnica para copiar sólo parte de una línea anterior o para combinar diferentes partes de otras líneas.

Nos hemos concentrado tanto en el tema de la edición que hemos descuidado lo relativo a la programación. Utilicemos algunas de las instrucciones que conocimos en el capítulo anterior. Escriba:

```
new<ENTER>
auto<ENTER>
10 ?"He aqui un punto y coma ";<ENTER>
20 ?"y mira lo que hace!"<ENTER>
30 <ESC>
```

Ejecute el programa. ¿Recuerda para qué sirve el punto y coma? Impide que el cursor salte a la línea siguiente. Edite la línea 1Ø para suprimir el punto y coma y vuelva a ejecutar el programa. Observará que el texto aparece ahora en dos líneas. Cuando el Amstrad se encuentra con una instrucción PRINT, empieza a escribir en una nueva línea, a no ser que la instrucción PRINT anterior haya terminado en punto y coma. Esto ocurre también con PRINT TAB y PRINT SPC:

```
new<ENTER>
auto<ENTER>
10 ?tab(9)"Esto es;"<ENTER>
20 ?tab(17)"el Amstrad."<ENTER>
30 <ESC>
```

Ejecute el programa y verá que todo el texto queda en la misma línea. Edite la línea 1Ø para suprimir el punto y coma. Observará que el texto se parte ahora en dos líneas.

Escriba el siguiente programa con la ayuda de la tecla COPY:

```
new<ENTER>
auto<ENTER>
10 ?tab(20);"A"<ENTER>
20 ?tab(19);"A A"<ENTER>
30 ?tab(18);"A";spc(3);"A"<ENTER>
40 ?tab(17);"A";spc(5);"A"<ENTER>
50 ?tab(16);"AAAAAAAAA"<ENTER>
60 ?tab(15);"A";spc(9);"A"<ENTER>
70 ?tab(14);"A";spc(11);"A"<ENTER>
80 <ESC>
```

Ejecute el programa y observe cómo hemos utilizado SPC. Esto es más cómodo que tener que contar cuidadosamente los espacios para escribir líneas como la siguiente:

```
60 PRINT TAB(15);"A          A"
```

La instrucción LOCATE

Utilizando la instrucción LOCATE se puede escribir en cualquier posición de la pantalla:

```
new<ENTER>
auto<ENTER>
10 mode 1<ENTER>
20 ?"Empezamos a escribir en (10,12):"<ENTER>
30 locate 10,12<ENTER>
40 ?"Aqui esta!"<ENTER>
50 <ESC>
```

La orden LOCATE de la línea 30 desplaza el cursor a la columna 10 de la línea 12 de la pantalla. Al ejecutarse la línea 40, el ordenador queda dispuesto para empezar a escribir en esa posición.

Ejecute el programa de nuevo, cambiando previamente la línea 10:

```
10 mode 0<ENTER>
```

El mensaje quedará escrito en la mitad derecha de la pantalla. El modo 0 sólo dispone de 20 columnas, por lo que la décima columna se encuentra en el centro. Si

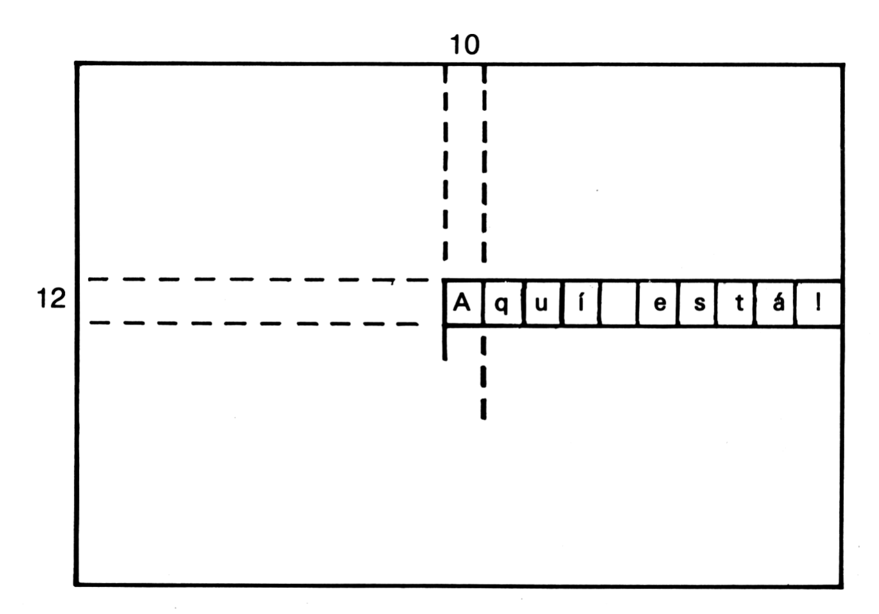


Figura 4. Control de la posición del cursor.

ejecutamos el programa en modo 2, el mensaje quedará mucho más cerca del borde izquierdo de la pantalla, puesto que este modo dispone de 8Ø columnas.

Desde luego, se puede utilizar SPC y TAB para acceder a otras posiciones, pero sin olvidar que estas instrucciones no permiten retroceder.

Ejercicios

Puede utilizar la tecla COPY para facilitar la programación de los siguientes ejercicios. Antes de abordar un programa nuevo, no olvide escribir new⟨ENTER⟩ para borrar el programa anterior.

1. Escriba un programa que dibuje lo siguiente:

```
***********
*         *
*         *
***********
```

2. Programe la realización de este dibujo en modo 1, comenzando en (1Ø, 2Ø):

```
*****
****
***
**
*
```

3. Utilizando TAB o SPC, programe el siguiente dibujo:

```
*****
 ****
  ***
   **
    *
```

4. Utilice PRINT TAB o PRINT SPC para programar esta representación en pantalla (en modo 1, centrada):

```
    Este         mensaje
      es         para
usuarios         de
         Amstrad
```

Utilización de variables

Escribamos un programa breve que haga que el ordenador escriba un mensaje:

```
new<ENTER>
auto<ENTER>
10 ?"Hola, amigo"<ENTER>
20 ?"Mi nombre es Amstrad"<ENTER>
30 ?"y creo que eres maravilloso!"<ENTER>
40 <ESC>
```

Si ejecutamos el programa, veremos que el ordenador nos escribe el mensaje, como era de esperar. Sin embargo, no se puede decir que este programa sea muy emocionante. Si volvemos a ejecutarlo, el mensaje que obtenemos es siempre el mismo. ¿No sería interesante que el ordenador se dirigiese a la persona que lo está utilizando por su propio nombre? Eso ya estaría mucho mejor. Escriba:

```
5 let nombre$="Mariano"<ENTER>
```

(ponga entre las comillas el nombre que desee). Modifique ahora la línea 10:

```
10 ?"Hola, amigo ";nombre$
```

Liste el programa y ejecútelo. En esta ocasión, el ordenador escribe un mensaje personalizado. No es que esto sea un gran progreso, pero lo cierto es que la diferencia entre los dos programas es importante.

La línea 5 dice:

```
5 let nombre$="Mariano"
```

La memoria del Amstrad se compone de miles de "células de memoria" que podemos considerar como una especie de casillero vacío. Cada casilla se puede utilizar para almacenar información. La línea 5 del programa pide al ordenador que busque un casillero vacío, lo denomine 'nombre$' y coloque en su interior la palabra 'Mariano'.

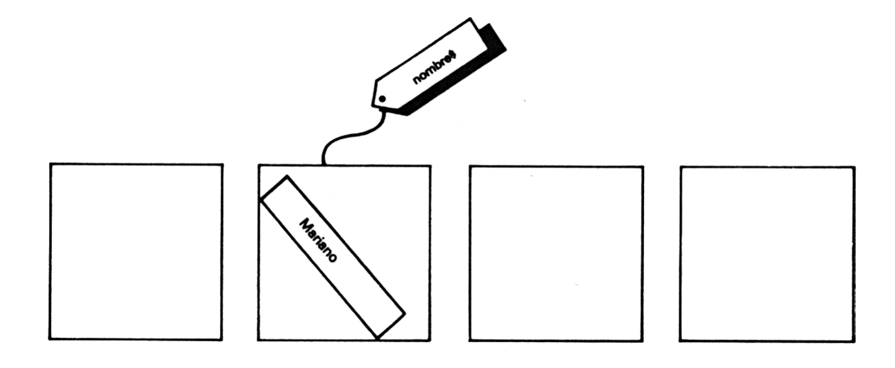


Figura 5. Cómo se almacena una cadena literal en la memoria.

El signo '=' de la línea 5 nos puede inducir a error si entendemos que '=' indica que dos cosas son exactamente iguales. En informática, una línea del estilo de la 5 se debe interpretar de esta forma:

> Toma la palabra 'Mariano' y almacénala en la dirección de la memoria en la que hemos puesto la etiqueta 'nombre$'.

Cada vez que hagamos referencia a nombre$ (como ocurre en la línea 10 del programa), el ordenador irá a la "casilla" etiquetada con 'nombre$' y tomará la palabra que encuentre allí. Esta palabra será utilizada después en la instrucción print, como podrá comprobar cuando ejecute el programa.

Si cambia el valor de nombre$ modificando la línea 5,

```
5 let nombre$="Atila el Huno"<ENTER>
```

y vuelve a ejecutar el programa, observará que el ordenador ha almacenado ahora 'Atila el Huno' en la zona de la memoria etiquetada con nombre$ y utiliza este nuevo nombre cuando ejecuta la instrucción de la línea 10. De hecho, en la casilla nombre$ se puede almacenar *cualquier cosa*; por eso decimos que nombre$ es una *variable.* Esto quiere decir que su valor varía y depende de lo que nosotros deseemos que sea. El signo '$' con que termina el nombre de la variable indica que nombre$ es una variable literal. Esto quiere decir que lo que se almacena en su interior es una cadena de caracteres (o sea, una cadena literal), que pueden ser alfabéticos, numéricos o signos de puntuación, o mezcla de estos tipos. Escriba:

```
5 let nombre$="123abc#$%"<ENTER>
```

y ejecute de nuevo el programa para comprobar que en nombre$ se puede almacenar cualquier cosa.

¿Tenemos que utilizar nombre$ necesariamente? Escriba:

```
5 let OtroNombreCualquiera$="John McEnro
e"<ENTER>
10 ?"Hola, amigo ";OtroNombreCualquiera$<ENTER>
```

y ejecute el programa. Al ordenador no le importa que demos a las variables nombres rebuscados. Sin embargo, conviene que al programar utilicemos nombres de variables que nos ayuden a recordar cuál es su contenido. Por ejemplo:

```
let marcadecoche$="Seat"
let titulodelibro="Las uvas de la ira"
let primerjugador$="Alfredito"
let ciudad$="Salamanca"
```

Nombres tales como 'eso$' o 'xyz$' no nos ayudarán a recordar para qué sirven cuando revisemos el programa al cabo de unos meses para tratar de corregirlo. No obstante, hay algunas limitaciones en cuanto a los nombres que podemos dar a las variables:

1. No pueden comenzar por un número, aunque sí pueden contener números en otras posiciones.
2. Sólo podemos usar caracteres alfabéticos o numéricos.
3. El nombre no puede constar de más de 40 caracteres.
4. No se pueden usar palabras clave (reservadas) tales como PRINT o LET.

Aunque se pueden utilizar indistintamente letras mayúsculas y minusculas, es aconsejable utilizar siempre minúsculas. Esto nos permite identificar las variables con más facilidad cuando listamos el programa, ya que el Amstrad convierte automáticamente a mayúsculas todas las palabras clave de BASIC, pero no los nombres de las variables.

El LET de la línea 5 del programa es *opcional*, lo que quiere decir que no es necesario incluirlo en la instrucción. Edite la línea 5 para suprimir el LET y verá que el ordenador ejecuta el programa igual que antes. Ahora que se ha familiarizado con LET, siento decirle que no lo volverá a ver en este libro, ya que su omisión nos ahorra bastantes pulsaciones.

Si tiene dudas sobre la posible validez de un nombre de variable, pruébelo en *modo inmediato* para ver cómo reacciona el Amstrad. Escriba:

```
estomarcha$="Bocadillo de calamares"<ENTER>
```

y el Amstrad emitirá el mensaje 'Ready', lo que indica que el nombre de esta variable es correcto; pero si escribe:

```
esano marcha$="10 huevos al plato"<ENTER>
```

obtendrá un mensaje de error, puesto que el nombre de una variable no puede contener espacios.

Cuando hemos asignado un valor a una variable, podemos ya utilizarla tantas veces como deseemos dentro del programa:

```
new<ENTER>
auto<ENTER>
10 let buenacomida$="helado"<ENTER>
20 let malacomida$="hormigas con chocola
te"<ENTER>
30 ?"Asi que le gusta el ";buenacomida$;
", verdad?"<ENTER>
40 ?"Yo, en cambio, prefiero ";malacomid
a$<ENTER>
50 ?buenacomida$;" es mejor para persona
s"<ENTER>
60 ?"Quisiera ver ";buenacomida$;" de ";
malacomida$<ENTER>
70 <ESC>
```

¿Qué ocurre si almacenamos una nueva cadena en una de las variables ya existentes? Escriba:

```
15 let buenacomida$="bizcocho"<ENTER>
```

y vuelva a ejecutar el programa. Verá que la cadena 'helado' se ha perdido al sustituirla por la cadena 'bizcocho'. La línea 15 pide al ordenador que almacene la palabra 'bizcocho' en la casilla 'buenacomida$' de la memoria, la descarta y la sustituye por 'bizcocho'.

Ejercicios

1. ¿Cuáles de los siguientes nombres son permisibles como nombres de variables?
   UNAPALABRAGRANDE$
   PresidentedeGobierno$
   15Moncloa$
   QuinceMoncloa$
   Quince-Moncloa$
   A$
   t$
   $
2. Corrija los errores del siguiente programa:

```
10 fecha actual$="15 de julio de 1985"
20 PRINT"Hoy estamos a";fecha actual$
```

3. Complete este programa para que al ejecutarlo escriba:
   Hola, Sr. Villanueva
   O puedo llamarle Marianito?

```
10 let apellido$="Villanueva"
20 let nombre$="Marianito"
```

4. Seleccione algunas palabras apropiadas que podamos almacenar en cadenas que pueda utilizar el ordenador para escribir una descripción del sheriff del condado. He aquí un ejemplo en el que las cadenas aparecen en mayúsculas para facilitar su identificación:

   Llevaba un SOMBRERO NEGRO en la CABEZA y se llamaba MUDD, más conocido por MUDD EL RAPIDO por su forma de manejar el REVOLVER y por el número de PERSONAS que había enviado a criar AMAPOLAS.

La instrucción INPUT

Nuestro programa nos ha servido para presentar las variables, pero no llega a ser mucho mejor que el programa original. Cada vez que lo ejecutamos, nos sigue dando el mismo mensaje, con el valor que hayamos asignado a nombre$ en la línea 5. Sin embargo, pretendíamos escribir un programa que diese un mensaje personalizado al usuario. Para cumplir ese objetivo necesitamos aprender para qué sirve la instrucción INPUT. Escriba:

```
new<ENTER>
auto<ENTER>
10 input"Como te llaman tus amigos";nomb
re$<ENTER>
20 ?"Espero que de encuentres bien, ";no
mbre$<ENTER>
30 <ESC>
```

Observe que no hemos asignado valor a nombre$ en ninguna parte del programa. ¿Cómo sabe el ordenador qué tiene que escribir en la línea 2Ø? Si ejecutamos el programa, el Amstrad escribirá:

```
Como te llaman tus amigos?█
```

Escriba su nombre (sin comillas) y pulse ⟨ENTER⟩. El ordenador capta su nombre, lo almacena en la variable nombre$ y lo utiliza cuando llega a la línea 2Ø. Ejecute el programa unas cuantas veces respondiendo con diferentes nombres. Hemos conseguido un programa más general que dará diferentes mensajes según quién lo utilice.

La instrucción INPUT permite que el ordenador obtenga información durante la ejecución del programa. El mensaje entrecomillado de la línea 1Ø se denomina *prompt* (que podríamos traducir por "apuntador" o "inductor") ya que nos indica que tenemos que escribir algo en el teclado, e incluso nos recuerda qué información espera el ordenador. El uso de este apuntador es opcional. También habríamos podido poner simplemente:

```
10 input nombre$
```

pero entonces el ordenador nos interrogaría con un frío y poco explícito signo '?' al ejecutar el programa. Ya hay demasiados programas de ese tipo por todas partes. Por lo tanto, incluyamos siempre un apuntador en las instrucciones INPUT.

El punto y coma del final del apuntador indica al Amstrad que sí queremos que escriba el '?' que él genera automáticamente en las instrucciones INPUT. Si lo sustituimos por una coma en la línea 1Ø, observaremos que el ordenador omite el '?'. En realidad, esto es sólo cuestión de estilo. Puede ser que usted prefiera una línea 1 más seria, sin signo de interrogación. Escriba:

```
1 input"Escriba su nombre: ",nombre$<ENTER>
run<ENTER>
```

y observe la diferencia.

Se puede utilizar una sóla instrucción INPUT para captar varias cadenas literales al mismo tiempo:

```
new<ENTER>
auto<ENTER>
10 input"Escriba su nombre y edad, separ
ados por una coma: ",nombre$,edad$<ENTER>
20 ?"Vamos, ";nombre$;". ";"No aparentas
 tener ";edad$;" años."<ENTER>
30 <ESC>
```

El Amstrad exige que los valores que introduzcamos queden separados por comas. De no hacerlo así, el ordenador emite el mensaje 'Redo from start' (repetir desde el principio). También se provoca el mismo mensaje si sólo se introduce una cadena cuando el ordenador espera dos, o si se introducen cuatro en lugar de tres. Esta es otra razón que hace aconsejable la inclusión del "apuntador" en las instrucciones INPUT, ya que nos recuerda cuántas entradas hacen falta y en qué consisten.

Ejercicios

1. Quizás haya recibido usted alguna vez una de esas temibles cartas "personalizadas" escritas por ordenador, en las que se le ofrece la oportunidad de ganar un premio en un concurso. Normalmente, se le invita al mismo tiempo a adquirir un libro muy útil, como puede ser el "Atlas Universal de los Gusanos". Escriba un programa que confeccione una carta de este tipo, después de captar por el teclado datos tales como nombre, dirección, fecha, etc. En el ejemplo siguiente, los datos captados por el teclado (con INPUT) son los que aparecen en mayúsculas.

Empresa de Libros Encuadernados en Piel
C/. Astucia, 31
66175 Guardalapala

25 de JULIO de 1984

Estimado SR. PÉREZ:

Estaba el otro día paseando junto al número 2Ø de la CALLE ARNULFO, cuando pensé en usted. El SR. PÉREZ es la persona más inteligente del barrio de los REYES GODOS, me dije. Estoy seguro de que tanto él como cualquier otra persona de la CALLE ARNULFO reconocerá el interés de invertir en "Inversiones Guardalapala" por sólo 2Ø.ØØØ pesetas, cuando ade-

más el SR. PÉREZ puede ganar también un viaje a la Empresa de Libros Encuadernados en Piel, con todos los gastos pagados. No se demore, SR. PÉREZ, sólo tiene siete días desde el 25 DE JULIO DE 1984 para beneficiarse de esta oportunidad única.

Atentamente,
Un buen amigo

2. Escriba un programa que escriba un "poema" después de captar por el teclado la terminación de tres de sus líneas. Por si le sirve de ayuda, le ofrecemos un ejemplo. Los datos captados con instrucciones INPUT son los que aquí figuran en mayúsculas:

   Erase una vez un tal MANOLO
   al que le gustaba COMER COMO A EL SOLO
   Un día se fue de su casa
   y se dedicó a ESPERAR A VER QUE PASA

3. Escriba un programa que le pregunte por su propio nombre y dirección y que escriba una tarjeta de visita como la siguiente:

```
.............................................
.        NOMBRE:          Mariano Villanueva .
.        DIRECCION:          Calle Arnulfo, 20 .
.                            Guardalapala      .
.                                              .
.............................................
```

La instrucción LINE INPUT

Seguro que este último ejercicio le ha dado problemas si en respuesta a una instrucción INPUT ha escrito una cadena que contuviese una coma. Hay ocasiones en que sucede esto. Por ejemplo, puede ocurrir que alguien desee escribir su nombre como 'Juan Rodríguez, S.J.'. En tal caso no sirve la instrucción INPUT normal, pues el Amstrad emitiría el mensaje de error 'Redo from start' al incluir una coma en la respuesta. Escriba:

```
new<ENTER>
auto<ENTER>
10 input"Escriba su profesion: ",profesi
on$<ENTER>
10 <ESC>
```

Al ejecutar este miniprograma observará que si respondemos con 'abogado, notario' obtenemos el mensaje 'Redo from start'. El ordenador se encuentra la coma y cree que la respuesta es 'abogado'. Como usted ha seguido escribiendo, "piensa" que su intención es introducir una segunda cadena y, naturalmente, protesta.

En previsión de accidentes de este tipo, debemos utilizar la instrucción LINE INPUT, que acepta una línea completa de texto, incluídas las comas. Modifique la línea 1Ø de la siguiente forma:

```
1Ø line input"Escriba su profesion: ",pr
ofesion$<ENTER>
```

y comprobará que ahora puede escribir *cualquier respuesta*, que será aceptada como entrada válida. Con cada LINE INPUT sólo se puede captar una cadena, porque ahora el ordenador ya no tiene forma de saber dónde termina una respuesta y comienza la siguiente. Otra cosa que hay que recordar LINE INPUT no genera automáticamente el signo '?' a que nos tenía acostumbrado INPUT. Si el signo de interrogación es necesario, tendrá que incluirlo explícitamente en la cadena apuntadora.

Variables numéricas

Deberemos utilizar variables literales siempre que queramos almacenar cadenas de caracteres alfabéticos o una mezcla de alfabéticos y numéricos. Parece lógico que podamos almacenar también números y, en efecto, podemos. Pero la forma de almacenarlos dependerá de lo que queramos hacer con ellos. Escriba:

```
new<ENTER>
auto<ENTER>
1Ø primernumero$="123"<ENTER>
2Ø segundonumero$="345"<ENTER>
3Ø ?"El primer numero es ";primernumero$<ENTER>
4Ø ?"El segundo numero es ";segundonumer
o$<ENTER>
5Ø <ESC>
```

Ejecute el programa. Funciona bien, ¿verdad? Si todo lo que pretendemos hacer con los números es escribirlos, no hay inconveniente en utilizar variables literales para almacenar sus valores. Pero veamos qué ocurre cuando intentamos realizar cálculos aritméticos. Modifique la línea 5Ø:

```
5Ø ?"Al multiplicarlos obtenemos";primer
numero$*segundonumero$<ENTER>
```

Ejecute el programa. El Amstrad emite el siguiente mensaje:

```
Type mismatch in 5Ø
```

(incongruencia de tipos en 5Ø). Un nuevo mensaje de error. Lo que indica es que el Amstrad no puede hacer cálculos aritméticos con cadenas literales. En realidad, esto es muy conveniente. Después de todo, primernumero$ podía muy bien haber

sido "martillos" y segundonumero$ podría haber sido "El océano Atlántico" y, claro, la multiplicación habría sido imposible. Antes de realizar cálculos aritméticos con variables, el Amstrad tiene que asegurarse de que está tratando con números. Por consiguiente, si tenemos números con los que hemos de realizar cálculos, éstos deberán ser almacenados en *variables numéricas*, no en variables literales.

Las variables numéricas son muy parecidas a las literales. La diferencia más evidente para nosotros es que carecen del signo '$' y que cuando les asignamos valores no utilizamos las comillas. Por ejemplo:

longitud=27
altura=7.6
distancia=38.45
gasolinagastada=5.5
temperatura=−10.5

son todos nombres de variables perfectamente razonables y aceptables. También son aceptables, aunque no muy útiles en un programa, los siguientes:

y33=5.6
A=75

Las variables numéricas pueden contener números enteros (sin decimales), tales como 27, o números decimales, tales como −10.5, pero no pueden contener cadenas literales. Compruébelo.

Si lista el programa original, verá que con algunos cambios sencillos podrá hacerlo funcionar. Sólo tiene que editar algunas líneas para conseguir que los números se almacenen en variables numéricas, y no como cadenas. Edite el programa para que tome la siguiente forma:

```
10 primernumero=123
20 segundonumero=345
30 ?"El primer numero es";primernumero
40 ?"El segundo numero es";segundonumero
50 ?"Al multiplicarlos obtenemos: ";prim
ernumero*segundonumero
```

Ejecute el programa. El efecto de la última línea debería ser el siguiente:

```
Al multiplicarlos obtenemos 42435
```

Pero añadamos algunas otras líneas. Escriba lo siguiente:

```
auto 60<ENTER>
60 ?"Al sumarlos obtenemos: ";primernume
ro+segundonumero<ENTER>
70 ?"Al dividirlos obtenemos: ";primernu
mero/segundonumero<ENTER>
80 ?"Al restarlos obtenemos: ";primernum
```

```
ero-segundonumero<ENTER>
90 <ESC>
run<ENTER>
```

En las líneas que van de la 5Ø a la 8Ø, el Amstrad toma los números de las "casillas" de la memoria y los utiliza para realizar los cálculos aritméticos. Podemos cambiar las líneas 1Ø y 2Ø para que el programa capte por el teclado (con INPUT) los valores de 'primernumero' y 'segundonumero':

```
10 input"Cual es el primer numero";prime
rnumero<ENTER>
20 input"Cual es el segundo numero";segu
ndonumero<ENTER>
```

El Amstrad efectuará ahora los cálculos con los números que el usuario introduzca.

Ejercicios

1. Usted va a visitar Rurislovaquia, donde la moneda es el ruri, que equivale a 3Ø5.4 pesetas. Escriba un programa para convertir pesetas en ruris después de preguntar al usuario cuántas pesetas quiere cambiar.
2. Escriba un programa que capte por el teclado el largo y el ancho de una alfombra y luego calcule y escriba su superficie.

Aritmética con variables

Aunque hasta ahora sólo hemos explicado unas cuantas instrucciones de BASIC, la utilización de las variables ampliará considerablemente sus horizontes. Por ejemplo, con sólo 3 líneas, usted puede escribir un programa para convertir kilómetros en millas:

```
new<ENTER>
auto<ENTER>
10 unamilla=1.6093<ENTER>
20 input"Cual es la distancia en kilomet
ros";distanciaenkilometros<ENTER>
30 ?"Eso equivale a";distanciaenkilometr
os/unamilla;"millas"<ENTER>
40 <ESC>
```

Pero ampliemos este programa. Su coche puede recorrer unas 3Ø millas con un galón de gasolina, y queremos saber cuántos galones vamos a necesitar para recorrer una determinada distancia, dada en kilómetros. En la línea 3Ø se calcula el número de millas. En vez de realizar este cálculo de nuevo en otra línea para después dividir

por 3Ø para averiguar cuántos galones necesitamos, es mucho más fácil conservar el resultado de la línea 3Ø almacenándolo en otra variable. Así pues, añada estas líneas a su programa:

```
10 unamilla=1.6093
20 INPUT"Cual es la distancia en kilomet
ros";distanciaenkilometros
25 distanciaenmillas=distanciaenkilometr
os/unamilla
30 PRINT"Eso equivale a";distanciaenkilo
metros/unamilla;"millas"
50 PRINT"Para recorrerlas se necesitan";
distanciaenmillas/30;"galones de gasolin
a."
```

La línea 25 toma el resultado de dividir la distancia en kilómetros por 1.6Ø93 y lo almacena en la variable 'distanciaenmillas', para luego volver a utilizarlo en la línea 5Ø para averiguar cuántos galones de gasolina necesitamos.

El precio de la gasolina es de £1.8Ø por galón, y usted desea saber cuánto costará el viaje. Vamos a tratar de generalizar este programa ahora, ya que quizás compre usted un nuevo coche dentro de poco y su consumo sea distinto. Además, es probable que el precio de la gasolina suba también. Tratemos todos estos números como variables:

```
10 unamilla=1.6093
12 millasporgalon=30
14 preciogasolina=1.8
20 INPUT"Cual es la distancia en kilomet
ros";distanciaenkilometros
25 distanciaenmillas=distanciaenkilometr
os/unamilla
30 PRINT"Eso equivale a";distanciaenkilo
metros/unamilla;"millas."
40 galonesparaviaje=distanciaenmillas/mi
llasporgalon
50 PRINT"Para recorrerlas se necesitan";
galonesparaviaje;"galones de gasolina,"
60 costoviaje=galonesparaviaje*preciogas
olina
70 PRINT"que cuestan";costoviaje;"libras
."
```

En las líneas 25, 4Ø y 6Ø utilizamos variables de valores conocidos para calcular los valores de nuevas variables, que son a su vez utilizadas en cálculos posteriores. Si usted cambia de coche o sube el precio de la gasolina, lo único que tiene que hacer es modificar las líneas 12 y 14 para asignar los nuevos valores y podrá seguir utili

zando el programa. ¿No será ésta una forma complicada de realizar esta tarea? Lo ideal sería disponer de un programa que nos preguntase cuántas millas por galón hace el coche y cuánto cuesta un galón de gasolina; de esa forma el programa podría ser utilizado por alguien que ni siquiera sepa qué es eso de "editar líneas". Cambie, pues, las líneas 12 y 14 para que el usuario pueda introducir los valores por el teclado, como ocurre en la línea 2Ø.

Ejercicios

1. En un programa anterior, el ordenador captaba los datos relativos al ancho y largo de una alfombra y calculaba el área. Amplíe el programa para que halle el precio de la alfombra cuando introduzcamos el coste del metro cuadrado.
2. El propietario de un restaurante de comida rápida acaba de comprar un ordenador y desea poder teclear el precio de los artículos y que el ordenador calcule el 12 por 1ØØ de IVA. Además, debe sumar el IVA al precio para dar el importe total.
3. El Club del Libro del Mes envía una carta mensual a sus socios en los siguientes términos:

Nombre del socio: Mariano Pérez	
Número de socio: 12345678	
Saldo anterior	1250 pesetas
El libro de este mes vale	+595
Total	1845
Su pago del mes pasado	−500
Saldo a nuestro favor	1345 pesetas

Escriba un programa que capte por el teclado el nombre del socio, su número, la cantidad que adeuda, el precio del libro del mes actual y el último pago mensual efectuado por el socio. El programa deberá escribir el detalle de los movimientos de la cuenta, tal como se indica más arriba.

¿Es 2=2+1?

Este extraño título sólo pretende recordarle que el signo '=' no siempre significa "igual a" en informática. Escriba lo siguiente:

```
new<ENTER>
auto<ENTER>
1Ø input"Escriba un numero: ",numero<ENTER>
2Ø numero=numero+1<ENTER>
3Ø ?"El numero de ha convertido en";nume
ro<ENTER>
4Ø <ESC>
```

Ejecute el programa y teclee un número cualquiera. Observará que el valor que aparece en la pantalla al final del programa es el número que usted introdujo más 1. ¿Cómo ha ocurrido esto?

Puede que parezca que la línea 2Ø carece de sentido, y así sería si el signo '=' significase "igual a". Revisemos el programa y veamos lo que le ocurre al número que usted introduce. En la línea 1Ø, el ordenador toma el valor que ha introducido y lo almacena en una "casilla" de la memoria, a la que pone la etiqueta 'numero':

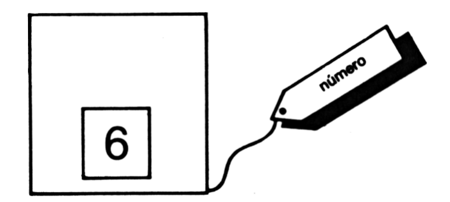


Figura 6. Así almacena el ordenador en su memoria el valor 6 que hemos introducido.

En la línea 2Ø lo que hacemos es decirle al ordenador:

> Saca el valor que hay en la casilla 'numero', súmale 1 y vuelve a colocar el resultado en la misma casilla.

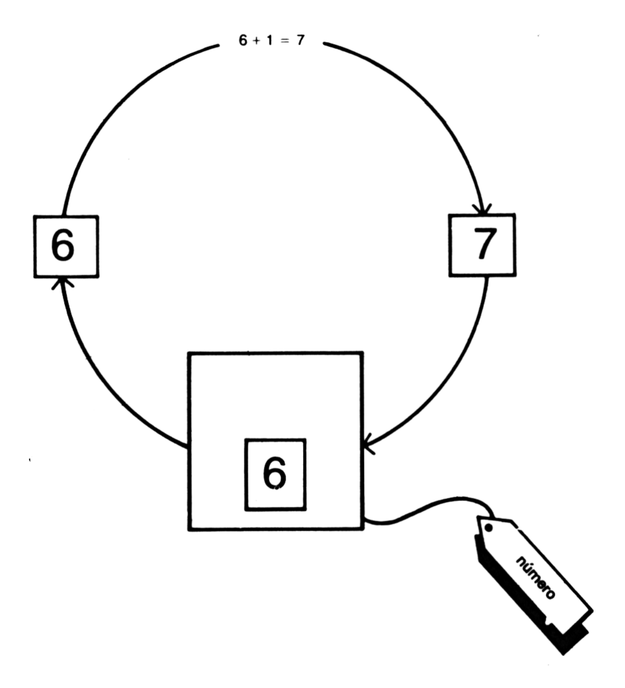


Figura 7. número=número+1. Así es cómo lo interpreta el Amstrad.

Análogamente ocurre en el ejemplo siguiente:

```
new<ENTER>
auto<ENTER>
10 input"Cuanto tenias ahorrado ayer";ah
orros<ENTER>
20 input"Cuanto has gastado hoy";gastos<ENTER>
30 ahorros=ahorros-gastos
40 ?"Tus ahorros son ahora";ahorros<ENTER>
50 <ESC>
```

Supongamos que sus ahorros eran de 60 pesetas y que usted ha gastado 20:

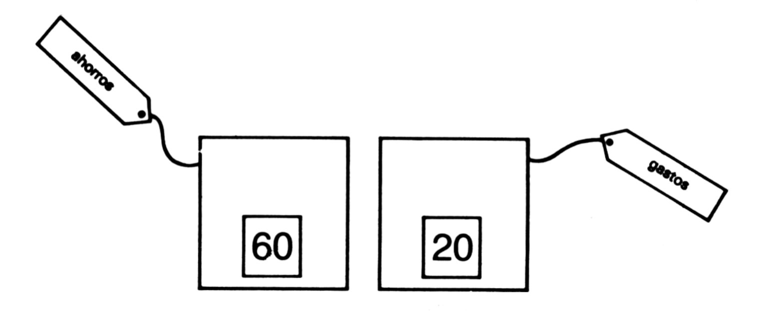


Figura 8. Ahorros y gastos.

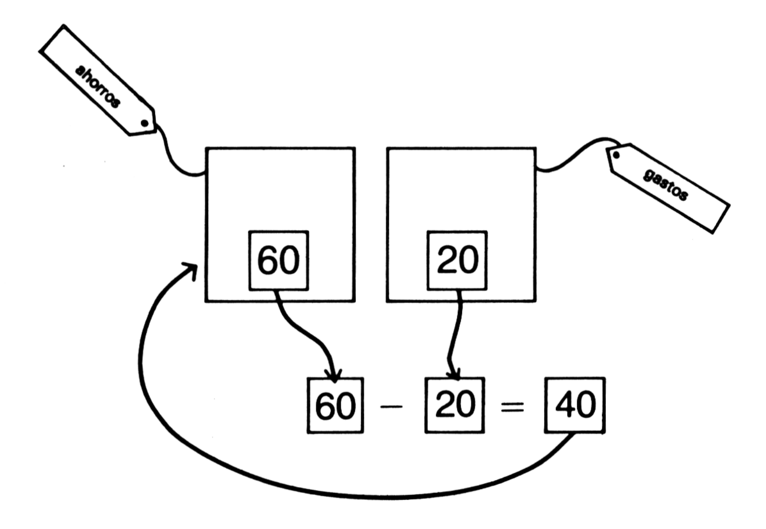


Figura 9. Ahorros y gastos.

La línea 3Ø hace que el ordenador tome el valor de los 'ahorros' (6Ø) y le reste los 'gastos' (2Ø); el resultado es 4Ø, que vuelve a quedar almacenado en 'ahorros'. (Ver figura 9.)

Éste es el valor que escribe la línea 4Ø. El valor almacenado en 'gastos' no se ve afectado.

Esta capacidad que poseen las variables, nos proporciona la forma de contar fácilmente hasta cierto valor:

```
new<ENTER>
auto<ENTER>
1Ø input"Introduzca un valor inicial: ",
valor<ENTER>
2Ø valor=valor+1<ENTER>
3Ø ?"Ahora es";valor<ENTER>
4Ø valor=valor+1<ENTER>
5Ø ?"Ahora es";valor<ENTER>
6Ø valor=valor+1<ENTER>
7Ø ?"Ahora es";valor<ENTER>
8Ø <ESC>
```

¿No le parece monótona esta repetición de líneas? Hay una forma más sencilla y corta de escribir este programa, pero tendremos que esperar hasta que hayamos avanzado un poco más para conocerla.

Ejercicios

1. El dueño del restaurante utiliza ahora su ordenador para sumar las facturas. Él introduce los precios de cuatro artículos y el ordenador los suma y escribe el total. También calcula los impuestos.

   La factura queda como sigue:

Artículo 1	95 pesetas
Artículo 2	12Ø
Artículo 3	35
Artículo 4	5Ø
Total	3ØØ
IVA	36
Importe total	336 pesetas

Este programa adolece de algunos defectos. Si en la factura sólo tenemos que incluir dos artículos, tendremos que introducir Ø como precio de los restantes, y si hay más de cuatro, tendremos que hacer otra factura. A veces los impuestos son cantidades con decimales, tales como Ø.48, y entonces el importe total contiene fracciones de peseta. Tanto el propietario del restaurante como usted tienen aún bastante que aprender.

Cómo grabar y cargar los programas

Quizás piense usted que ya es capaz de escribir programas que vale la pena grabar en cinta para poder utilizarlos en el futuro. Si es así, consulte la página F1.13 de la Guía del Usuario, donde se explica cómo hacerlo. Las páginas F1.10 a F1.12 le indican cómo puede volver a cargar el programa en la memoria desde la cinta.

Es aconsejable hacer constar claramente en la etiqueta de la cinta el nombre del programa o programas que contiene, ya que de otro modo acabará con un montón de cintas y nunca sabrá cuál de ellas contiene su última obra maestra, ni con qué título la había grabado.

¡Adelante con la programación!

Estos dos primeros capítulos sólo han servido para que el lector se familiarizase un poco con la informática. Hemos hablado de la RAM y la ROM, de las variables, que son fundamentales para poder manejar los datos, y hemos presentado algunas de las palabras clave, PRINT, LET e INPUT, utilizándolas en pequeños programas.

En los capítulos que siguen aceleraremos el ritmo, así que no intente leerse todo el libro en un día. No pase nada por alto, ya que los programas de algunos capítulos utilizan instrucciones de BASIC que se explican en los anteriores.

A partir de ahora ya no le recordaremos que debe pulsar ⟨ENTER⟩ cada vez que termine de escribir una línea de programa o una orden directa. Además, le dejaremos en libertad para utilizar AUTO o RENUM, así como las demás ayudas de edición del Amstrad, como usted estime oportuno.

Por último, le recordaremos que existen diferentes formas de escribir los programas para que realicen una misma tarea concreta. Los ejercicios propuestos no tienen una solución única, sino varias; por eso no hemos creído conveniente incluir un apéndice con la lista de soluciones. Si su programa funciona, la solución es correcta. Más adelante veremos cómo podemos simplificar la elaboración de programas largos mediante sistemas de planificación. Pero, por el momento, utilice su propia intuición y páselo bien.

3

Dibujos

LA PANTALLA GRÁFICA

En el capítulo 2 vimos que el Amstrad puede situar el texto en cualquier lugar de la pantalla utilizando las *coordenadas de texto*. El ordenador también puede dibujar líneas, pero tenemos que indicarle dónde comienzan y acaban éstas. La pantalla gráfica se divide en 64Ø puntos en sentido horizontal y 4ØØ en vertical.

Podemos identificar la posición de cualquier punto de la pantalla especificando sus coordenadas.

La posición del punto de la figura es (2ØØ, 3ØØ). El primer número es la *coordenada X* del punto (posición en el eje horizontal); el segundo es la *coordenada Y* (posición en el eje vertical).

Observe que las *coordenadas gráficas* se miden a partir del *extremo inferior izquierdo* de la pantalla, y que el punto situado en esa posición tiene las coordenadas

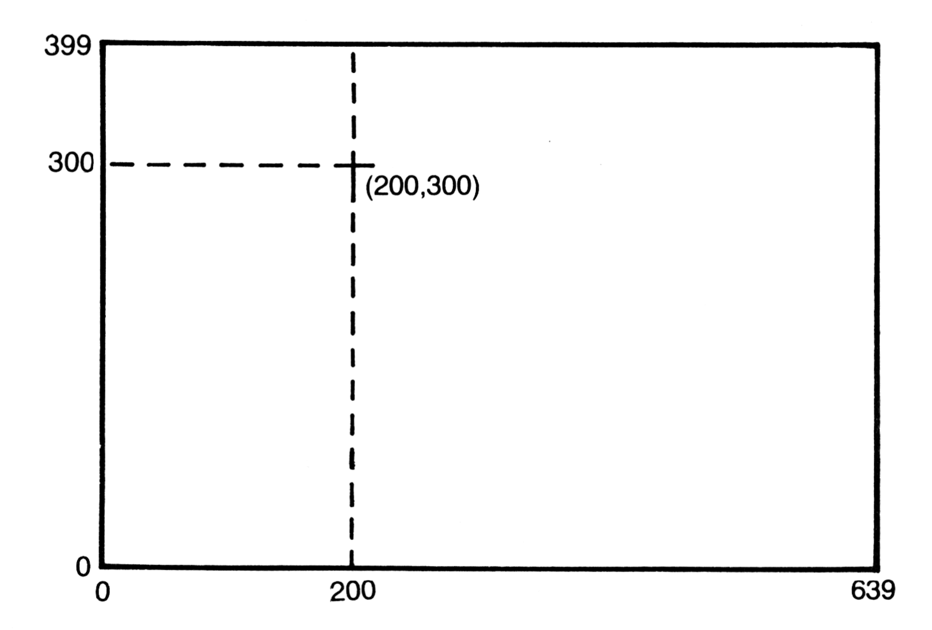


Figura 10. El punto (2ØØ, 3ØØ) en la pantalla gráfica.

(Ø, Ø). Esto puede prestarse a confusión, ya que las coordenadas de texto funcionan de modo completamente diferente: se miden a partir del extremo superior izquierdo, el cual tiene las coordenadas (1, 1). Observe también que la numeración de los puntos empieza por Ø, y que por lo tanto el extremo superior derecho de la pantalla gráfica tiene las coordenadas (639, 399), *no* las coordenadas (64Ø, 4ØØ) como cabría pensar. Pruebe el siguiente programa:

```
10 MODE 1
20 MOVE 124,156
30 DRAW 300,300
40 DRAW 200,400
50 DRAW 124,156
```

Cuando usamos las órdenes de gráficos del Amstrad, utilizamos el *cursor gráfico* para trazar líneas sobre la pantalla. Normalmente, el cursor gráfico y el de texto permanecen juntos, pero tan pronto como ejecutamos una orden gráfica, entra en acción el cursor invisible de gráficos.

La orden MOVE de la línea 2Ø hace que el cursor de gráficos se desplace invisiblemente al punto (124, 156). La orden DRAW hace que el cursor se desplace desde la posición (124, 156) hasta el punto de coordenadas (3ØØ, 3ØØ) trazando una recta entre ambos puntos. Las restantes ordenes DRAW de las líneas 4Ø y 5Ø trazan los otros dos lados del triángulo.

En resumen, MOVE x,y hace que el cursor de gráficos se desplace hasta el punto x, y sin dibujar a su paso, mientras que DRAW x,y dibuja una recta desde el último punto visitado (como resultado de una orden MOVE o DRAW) hasta punto x, y.

Los diferentes modos de pantalla

Aunque la pantalla de gráficos conste de 64Ø puntos horizontales y 4ØØ verticales, lo cierto es que el Amstrad no puede dintinguirlos todos ellos. En el capítulo 1 vimos que hay tres modos de pantalla, y que en cada modo el tamaño de los caracteres es diferente. Sin embargo, la pantalla gráfica es la misma para los tres modos, aunque el Amstrad distingue mejor unos puntos de otros según el modo que se utilice. Ejecute de nuevo el programa anterior, después de haber cambiado la línea 1Ø como sigue:

```
10 MODE 0
```

Observará que el dibujo sigue teniendo la misma forma, pero con las líneas más gruesas. Pruebe ahora:

```
10 MODE 2
```

Ahora las líneas se han hecho muy finas. El modo 2 es el denominado *modo de alta resolución*, ya que en él el ordenador puede distinguir 64Ø puntos en horizontal

y 200 en vertical; a esto se debe el que cuando usamos la instrucción DRAW las rectas resultantes sean tan finas. En modo 2, el Amstrad no puede establecer diferencias entre puntos que se encuentren muy juntos en el plano vertical. Por ejemplo, no puede distinguir entre el punto (10, 10) y el (10, 11). Los tres modos, 0, 1 y 2, dan la misma resolución vertical, pero la resolución horizontal del 2 es muy superior. Escriba:

```
10 MODE 1
```

y vuelva a ejecutar el programa. El modo 1 es el de *resolución media*; en este modo el Amstrad sólo puede distinguir 320 puntos en horizontal. Esto quiere decir que, por ejemplo, los puntos (200, 300), (200, 301), (201, 300) y (201, 301) son tratados como si fuesen uno solo. Escriba ahora:

```
10 MODE 0
```

y ejecute el programa por tercera vez. El modo 0 es el de *baja resolución* y sólo puede discriminar 160 puntos en el eje horizontal.

Quizá se pregunte usted por qué querríamos utilizar un modo de pantalla que produce dibujos tan toscos, cuando se puede usar el modo 2 de alta resolución. La razón es la siguiente: aunque en modo 0 la resolución sea tan baja, puede visualizar hasta 16 colores diferentes al mismo tiempo.

Esto no es posible con los modos 1 y 2.

	Resolución gráfica	Número de colores visualizables simultáneamente
Modo 0	160×200	16
Modo 1	320×200	4
Modo 2	640×200	2

Figura 11. Grados de resolución y número de colores disponibles en los diversos modos.

No olvidemos que la memoria del ordenador tiene sus límites. En la RAM sólo se puede almacenar cierta cantidad de información acerca de la pantalla. En informática todo tiene un precio. Podemos utilizar la RAM para almacenar detalles de muchos puntos en dos colores, menos detalles en 4 colores y muchos menos en 16 colores. El Amstrad nos permite elegir lo que más convenga en cada aplicación concreta.

Ejercicios

1. Escriba un pequeño programa que dibuje un rectángulo utilizando las instrucciones MOVE y DRAW.
2. Dibuje la cabeza de un robot utilizando las instrucciones MOVE y DRAW:

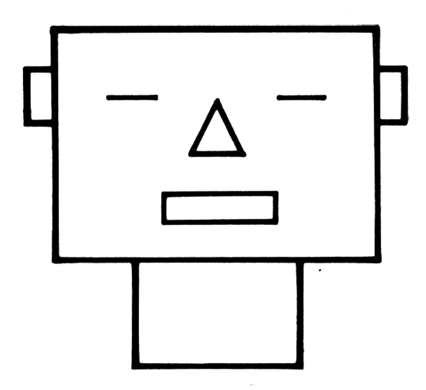


Figura 12. La cabeza de un robot.

3. Escriba un programa que dibuje los siguientes triángulos:

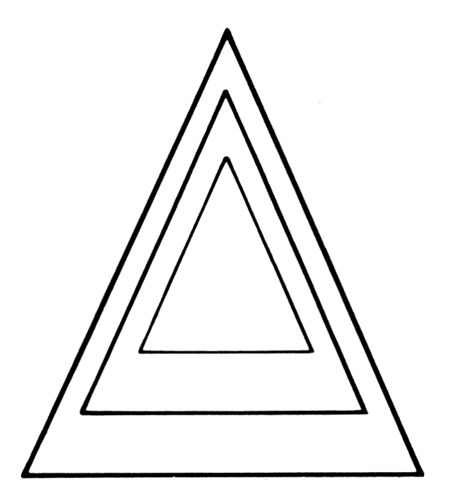


Figura 13. Triángulos.

La instrucción PLOT

El Amstrad también puede representar puntos aislados en la pantalla, aunque en modo 2 un punto solo es difícil de ver. El siguiente programa dibuja 6 puntos diferentes:

```
10 MODE 0
20 PLOT 160,200
30 PLOT 320,200
40 PLOT 324,200
50 PLOT 328,200
60 PLOT 332,200
70 PLOT 480,200
```

En modo 0 la resolución es tan baja, que los cuatro puntos dibujados por las líneas 30 a 60 se funden para formar un segmento de recta. En modo 1 se pueden distinguir todos los puntos, mientras que en modo 2 los puntos son tan finos que quizás no pueda usted ni verlos.

PLOT funciona del mismo modo que MOVE y DRAW. Tiene que ir seguido de las coordenadas X e Y de los puntos que deseamos representar. En realidad, cada "punto" se compone de cierto número de puntos de luz, porque ni siquiera el modo 2 es lo suficientemente preciso como para distinguir todos los puntos físicos de la pantalla. El grupo de puntos más pequeño que se puede identificar en la pantalla en los diferentes modos es lo que se denomina un *pixel*.

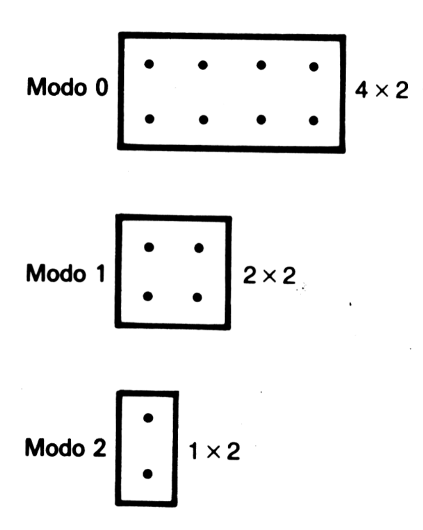


Figura 14. Tamaños de los pixels en los diversos modos de pantalla.

Variables y programas gráficos

El siguiente programa dibuja el contorno de una casa, incluido el techo:

```
10 MODE 1
20 REM dibuja un rectangulo que es la fachada de
   la casa
30 MOVE 120,100
40 DRAW 500,100
50 DRAW 500,300
60 DRAW 120,300
70 DRAW 120,100
80 MOVE 120,300
90 DRAW 185,380
100 DRAW 435,380
110 DRAW 500,300
```

Este programa contiene una instrucción nueva, REM, abreviatura de *REMmark*, que en inglés significa "nota" u "observación". Permite hacer anotaciones que ayudan a seguir el hilo de los programas. Como nuestros programas se van haciendo cada vez más largos, las incluiremos para explicar sus diferentes partes. Cuando el Amstrad se encuentra una instrucción REM, ignora absolutamente todo el resto de la línea. Esta instrucción no sirve más que para facilitar la tarea del programador.

Por otra parte, los programas largos presentan otro problema ya que al listarlos no caben enteros en la pantalla. No obstante, podemos listar cualquier parte de un programa tecleando, por ejemplo,

```
list 10-30
```

Esta orden lista las líneas comprendidas entre la 1Ø y la 3Ø, ambas inclusive.

Podemos hacer los programas más interesantes utilizando variables, cuyos valores se pueden cambiar fácilmente:

```
10 MODE 1
20 REM coordenadas de la fachada
30 casaizquierda=120
40 casaabajo=100
50 casaderecha=500
60 casaarriba=300
70 REM dibujar la fachada
80 MOVE casaizquierda,casaabajo
90 DRAW casaderecha,casaabajo
100 DRAW casaderecha,casaarriba
110 DRAW casaizquierda,casaarriba
120 DRAW casaizquierda,casaabajo
```

```
130 REM coordenadas del tejado
140 tejadoizquierda=185
150 tejadoarriba=380
160 tejadoderecha=435
170 REM dibujar el tejado
180 MOVE casaizquierda,casaarriba
190 DRAW tejadoizquierda,tejadoarriba
200 DRAW tejadoderecha,tejadoarriba
210 DRAW casaderecha,casaarriba
```

Observe que, puesto que muchos pares de coordenadas se utilizan dos veces, no necesitamos tantas variables como en principio sería de temer. Ahora podemes reducir la altura de la casa sin más que cambiar la línea 60 para darle una coordenada Y menor; por ejemplo, 200. O bien podemos subir el tejado modificando la línea 50. Esto nos demuestra la gran potencia e interés que tienen las variables. Si hubié-

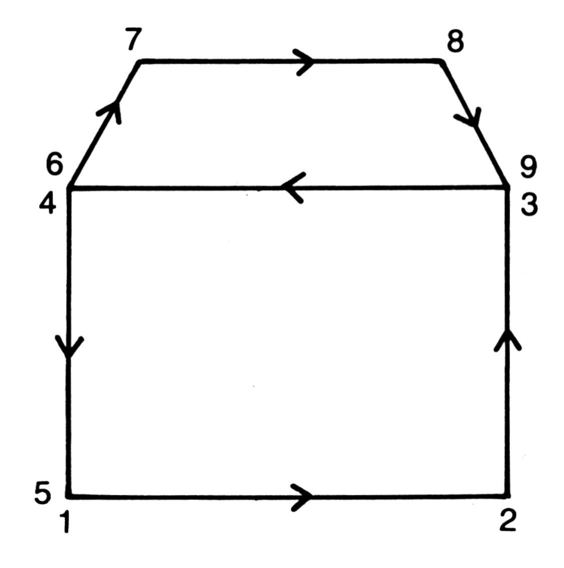


Figura 15. Casa.

semos dibujado la casa utilizando números concretos, habríamos tenido que cambiar muchas líneas para probar las mismas modificaciones. ¿Cómo podemos estrechar la casa? ¿Por qué tenemos que cambiar dos variables en vez de una? ¿Qué hará el Amstrad si le damos una coordenada demasiado grande, como podría ser 500 en la línea 150?

Cuando trabajamos con gráficos, hay muchas formas de producir el mismo efecto. Vale la pena que dediquemos algún tiempo a pensar el dibujo que queremos obtener, pues podemos acortar el programa considerablemente si pasamos por las esquinas o ángulos del dibujo en el orden correcto. La figura 15 muestra el orden en que se ha dibujado la casa.

Ejercicios

1. El programa con que hemos dibujado la casa emplea 9 instrucciones MOVE y DRAW. ¿Podría acortarlo usted a 8 trazando las líneas en un orden más eficaz?
2. Escriba un programa que realice el siguiente dibujo:

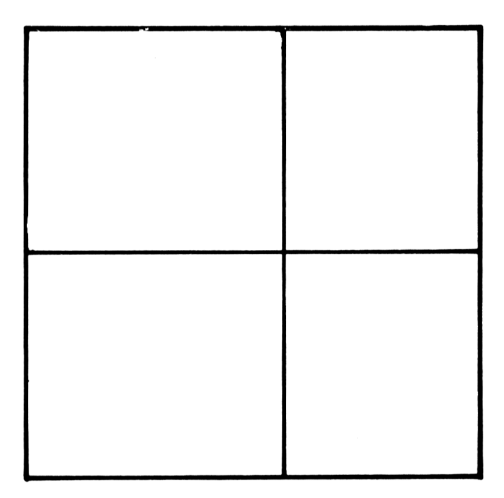


Figura 16. Ventana.

La instrucción INPUT en los gráficos

La utilización de la instrucción INPUT nos causa problemas, como podrá ver si borra las líneas 30 a 60 y las sustituye por:

```
30 INPUT "Coordenada x del lado izquierdo ";casa
   izquierda
40 INPUT "Coordenada x del lado derecho ";casade
   recha
50 INPUT "Coordenada y del suelo ";casaabajo
60 INPUT "Coordenada y de la base del tejado ";c
   asaarriba
```

Los apuntadores y los valores tecleados estropean el dibujo. Para evitar que esto ocurra, podemos pedir al Amstrad que dedique a los textos sólo una parte de la pantalla. Esto se logra fácilmente con la instrucción WINDOW:

```
10 MODE 1
20 REM coordenadas de la fachada
25 WINDOW 1,40,20,25
30 INPUT "Coordenada x del lado izquierdo ";casa
   izquierda
40 INPUT "Coordenada x del lado derecho ";casade
   recha
50 INPUT "Coordenada y del suelo ";casaabajo
60 INPUT "Coordenada y de la base del tejado ";c
   asaarriba
70 REM dibujar la fachada
80 MOVE casaizquierda,casaabajo
90 DRAW casaderecha,casaabajo
100 DRAW casaderecha,casaarriba
110 DRAW casaizquierda,casaarriba
120 DRAW casaizquierda,casaabajo
130 REM coordenadas del tejado
140 tejadoizquierda=185
150 tejadoarriba=380
160 tejadoderecha=435
170 REM dibujar el tejado
180 MOVE casaizquierda,casaarriba
190 DRAW tejadoizquierda,tejadoarriba
200 DRAW tejadoderecha,tejadoarriba
210 DRAW casaderecha,casaarriba
```

Podemos elegir la posición y tamaño de la ventana a la que relegamos el texto; para ello hemos de especificar los números de columna y de fila de sus cuatro bordes en el siguiente orden: columna izquierda, columna derecha, fila superior, fila inferior. Como estamos definiendo una ventana *de texto*, tendremos que utilizar coordenadas *de texto* para describirla.

La ventana continúa activa aunque haya terminado el programa, como se puede comprobar listándolo. Para que todo vuelva a la normalidad, ejecute una orden MODE. Si lo desea, también puede completar el programa cambiando las líneas 140 a 160 de forma que las coordenadas del tejado se introduzcan también por el teclado.

También podríamos poner la ventana en la parte superior de la pantalla:

```
25 WINDOW 1,40,1,2
```

Trate de ajustar la ventana para que quede a la derecha del dibujo.

Quizá se haya dado cuenta de que si el tamaño de la ventana no es el adecuado,

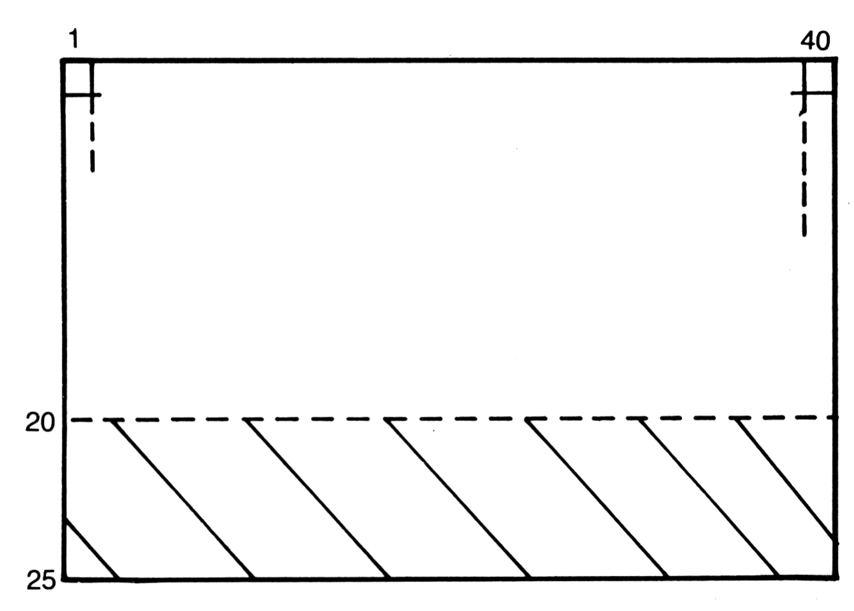


Figura 17. La zona sombreada representa la ventana de texto que se define con WINDOW 1,4Ø,2Ø,25.

el texto puede ser difícil de seguir. Para evitar problemas, vale la pena dedicar algún tiempo a planificar qué parte de la pantalla vamos a utilizar para texto y qué parte para gráficos.

Observe que al establecer una ventana de texto no afectamos en nada a la pantalla gráfica. Las líneas que dibujemos pueden superponerse al texto si la ventana de texto no está en el lugar correcto; por ejemplo, si la hemos puesto en el centro de la pantalla.

Ejercicios

1. Escriba un programa que capte tres pares de coordenadas y dibuje con ellos un triángulo.
2. Escriba un programa que capte cuatro pares de coordenadas y dibuje con ellos una flecha.

UN TOQUE DE COLOR

Al conectarlo, el Amstrad selecciona automáticamente el color amarillo para el texto y los gráficos y el azul para el fondo. El total de colores diferentes es de 27, aunque algunos de ellos no son fáciles de distinguir (en particular, por supuesto, cuando se está utilizando el ordenador con un monitor monocromático). Cada color se identifica por un número, denominado número de INK (tinta). Véase la figura 18.

Número de tinta	Color
0	Negro
1	Azul
2	Azul intenso
3	Rojo
4	Magenta (rojo + azul)
5	Malva
6	Rojo intenso
7	Morado
8	Magenta intenso
9	Verde
10	Cyan
11	Azul celeste
12	Amarillo
13	Blanco
14	Azul pastel
15	Anaranjado
16	Rosado
17	Magenta pastel
18	Verde intenso
19	Verde marino
20	Cyan intenso
21	Verde lima
22	Verde pastel
23	Cyan pastel
24	Amarillo intenso (dorado)
25	Amarillo pastel
26	Blanco intenso

Figura 18. Los 27 colores de tinta que se pueden utilizar en el Amstrad.

Al llegar a este punto, conviene advertir, por si el lector no lo hubiera observado, que el ordenador no utiliza toda la pantalla al escribir o al dibujar gráficos. El Amstrad trabaja en un gran rectángulo, alrededor del cual queda una zona que no puede utilizar: el *borde*. Véase la figura 19.

Aunque el Amstrad no utiliza este borde, el usuario puede controlar su color. El ordenador tiene que dedicar buena parte de su memoria a almacenar la información sobre el contenido de la pantalla (qué letras hay en ella, en qué posiciones, forma y color de las líneas, etc). Aunque no lo parezca, la imagen se está actualizando continuamente; de hecho, el Amstrad la redibuja 50 veces por segundo, consultando cada vez la zona de memoria dedicada a la pantalla.

El borde es otra cosa. Como el ordenador no lo utiliza para dibujar ni escribir, le basta con controlar su color. Éste es el único dato que el ordenador necesita para redibujar el borde.

Debido a esta diferencia, el borde recibe un tratamiento distinto en lo que a colo-

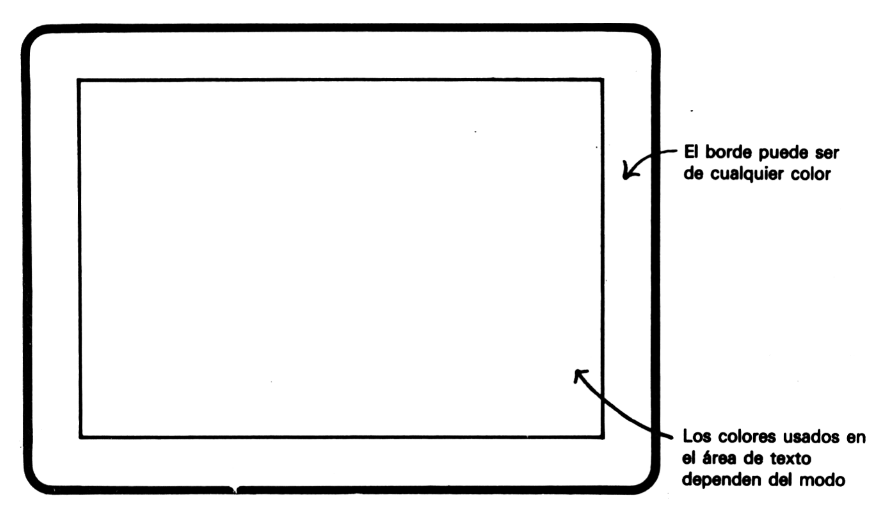


Figura 19. El borde en el monitor o televisor.

res respecta. Al borde se le puede asignar *cualquier* color en *cualquier* modo, sin restricción alguna. En modo 2 sólo se pueden visualizar 2 colores al mismo tiempo dentro del rectángulo principal de la pantalla, pero en cambio el borde puede tener *cualquier* color que deseemos. Escriba:

```
MODE 2
BORDER Ø
```

Si consulta la figura 18, verá que el Ø es el número de tinta del color negro. Con la orden BORDER Ø estamos haciendo que el Amstrad que exhiba un borde negro. Dé diferentes colores al borde. Puede utilizar cualquier número del Ø al 26, es decir, 27 en total. Por ejemplo, BORDER 26 establece un borde de color blanco.

En los modos Ø y 1 el color del borde se elige de la misma manera. Los cambios de modo de pantalla no afectan al color del borde. Al encender el ordenador se establece automáticamente el color 1 (azul). Usted puede incluir una orden BORDER en cualquiera de los programas que hemos escrito hasta ahora. No afectará al funcionamiento del programa, sino sólo a la estética de la pantalla.

Los colores de PEN y PAPER

Como era de esperar, también podemos modificar los colores con los que se forma el rectángulo principal de la pantalla. Aquí es donde se aprecian las limitaciones impuestas por la RAM, relativas el número de colores que se pueden visualizar simultáneamente en función del grado de resolución.

Para cambiar el color que el Amstrad utiliza para "escribir" necesitamos una orden PEN (pluma). Escriba:

```
MODE 0
PEN 4
```

De la lista de la figura 18 el lector podría esperar que esta orden produjese caracteres de color magenta, pero ya hemos advertido que los colores de la pantalla no se controlan de la misma manera que los del borde. Con PEN 4 no se seleccniona el color número 4, sino la pluma número 4. Podemos imaginar que el Amstrad trabaja con varias plumas, que podemos cargar en diferentes tinteros. La orden PEN selecciona una de las plumas, mientras que cada orden INK carga una pluma en un tintero especificado. En la figura 20 se da una lista de las tintas con que está cargada cada una de las 16 plumas posibles en el momento de encender el ordenador. Como se puede ver, la pluma número 4 está cargada con tinta de color 26 (blanco). Si ahora escribimos

```
PEN 1
```

seleccionamos una pluma que está cargada con tinta negra. Incluso podemos seleccionar una pluma cargada con *dos* colores mediante la orden:

```
PEN 14
```

que produce un efecto de parpadeo entre los colores azul y amarillo.

Número de PEN o PAPER	Modo Ø	Modo 1	Modo 2
Ø	1	1	1
1	24	24	24
2	2Ø	2Ø	1
3	6	6	24
4	26	1	1
5	Ø	24	24
6	2	2Ø	1
7	8	6	24
8	1Ø	1	1
9	12	24	24
1Ø	14	2Ø	1
11	16	6	24
12	18	1	1
13	22	24	24
14	1/24	2Ø	1
15	16/11	6	24

Figura 20. Colores asignados a las distintas plumas (PEN) y papeles (PAPER) en los distintos modos antes de modificarlos con instrucciones INK.

Como hemos dicho, el máximo número de plumas disponibles es 16, aunque no todas ellas se pueden utilizar simultáneamente en todos los modos. Observe en la figura 20 que una pluma que en modo Ø escriba con un color determinado puede hacerlo con un color diferente en los modos 1 y 2, de modo que un programa gráfico que funcione correctamente en modo Ø puede no producir ningún efecto en modo 2.

Como puede observar en la figura 20, las 16 plumas del modo 2 no sirven de mucho, ya que 8 están cargadas con tinta amarilla y las otras 8 con tinta azul.

La instrucción INK cambia el color de la tinta con que está cargada una pluma determinada. Por ejemplo, con la instrucción INK 1,15, la pluma 1, que originalmente estaba cargada con color 24 (amarillo), se carga con color 15 (anaranjado).

Para ver, por ejemplo, el color rosa, pase a modo Ø y escriba lo siguiente:

```
PEN 11
```

Cuando nos ponemos a jugar con los colores, es fácil que nos encontremos con que no podemos ver lo que estamos tecleando, bien porque los colores de pluma y de papel sean iguales o porque no sean suficientemente contrastados. Cuando se encuentre en esta situación, puede teclear una orden PEN aunque no vea aparecer en la pantalla las letras que escribe. Si está demasiado confuso, siempre puede *reinicializar* la máquina accionando las teclas ⟨CTRL⟩ y ⟨SHIFT⟩ y pulsando, antes de soltarlas, la tecla ⟨ESC⟩. Tenga en cuenta, no obstante, que la reinicialización borra completamente la RAM, con la consiguiente pérdida del programa.

Dicho sea de paso, en cualquier momento puede ocurrir que ejecutemos un programa que el ordenador no puede acabar. Siempre que parezca que el ordenador se ha "atascado", se puede interrumpir el programa pulsando la tecla ⟨ESC⟩. Con una pulsación, el programa se detiene momentáneamente; si a continuación se pulsa cualquier otra tecla, el programa se reanuda. Pulsando dos veces seguidas la tecla ⟨ESC⟩, el programa se interrumpe definitivamente. (Para reanudarlo se puede ejecutar la orden CONT, a condición de que durante la interrupción no se haya modificado ninguna línea si introducido otras nuevas.)

También se puede cambiar el color del fondo, utilizando para ello la orden PAPER, que funciona de forma similar a la PEN. Reinicialice el ordenador con ⟨CTRL⟩ ⟨SHIFT⟩ ⟨ESC⟩. Pase al modo Ø y escriba:

```
PAPER 3
```

Observará que los caracteres que aparecen a continuación están sobre fondo rojo. Puede cambiar toda la zona interior de la pantalla a este nuevo color escribiendo:

```
CLS
```

En modo Ø el papel (fondo) puede ser de los mismos 16 colores que las plumas. Así, lo mismo que PEN 14 nos daba caracteres de color que alterna entre el azul y el amarillo, PAPER 14 nos da un fondo de los mismos colores. La figura 2Ø le ayudará a seleccionar los colores de pluma y papel. Por ejemplo, para obtener caracteres rojos sobre fondo blanco en modo Ø, escriba:

```
PEN 3
PAPER 4
CLS
```

Naturalmente, las órdenes PEN y PAPER se pueden utilizar también incluidas en líneas de programa:

```
10 MODE 0
20 LOCATE 2,7
30 PEN 3
40 PAPER 5
50 PRINT "Rojo sobre negro"
60 LOCATE 2,13
70 PEN 6
80 PAPER 3
90 PRINT "Azul sobre rojo"
100 LOCATE 2,19
110 PEN 5
120 PAPER 6
130 PRINT "Negro sobre azul"
140 REM devolver PEN y PAPER a lo normal
150 PEN 1
160 PAPER 0
```

Si ejecuta este programa en los otros dos modos, cambiando la línea 10, observará que se obtiene resultados extraños, puesto que las plumas están cargadas con tintas diferentes en los otros modos. También es posible utilizar variables para seleccionar plumas y papeles:

```
10 MODE 0
20 plumarojaenmodo0=3
30 papelnegroenmodo0=5
40 PEN plumarojaenmodo0
50 PAPER papelnegroenmodo0
60 CLS
70 LOCATE 8,12
80 PRINT "Hecho!"
90 REM devolver PEN y PAPER a lo normal
100 PEN 1
110 PAPER 0
```

Las dos últimas líneas de este programa hacen que el ordenador vuelva a los colores normales, para que no se encuentre usted con una combinación ilegible como, por ejemplo, amarillo sobre blanco.

Ejercicios

1. Cambie el programa de la página anterior para que escriba los tres mensajes en verde sobre blanco, rojo sobre amarillo y blanco sobre negro. Ponga el borde color de cyan.
2. También se puede escribir cada letra de una palabra con un color diferente. Utilice la instrucción LOCATE para desplazarse hasta el punto adecuado y cambie las plumas antes de escribir cada letra. Escriba las palabras ARCO IRIS en modo Ø aplicando un color diferente a cada letra.
3. Escriba un programa que capte por el teclado los números de PEN y PAPER que se han de utilizar, que borre la pantalla y que escriba la frase "Nuevos colores" en el centro. (Los números de PEN y PAPER se deberán captar mediante instrucciones INPUT referidas a variables numéricas, no a variables literales.)

Gráficos y colores

Dibujar en color es muy fácil con el Amstrad. Ya hemos utilizado las órdenes MOVE y DRAW en algunos programas. Tal como las conocemos, estas órdenes dibujan rectas con la pluma número 1, cualquiera que sea el modo en que estemos. Así pues, todos los programas de gráficos que hemos visto hasta ahora han producido líneas trazadas con PEN 1. PEN 1 contiene la tinta número 24 en todos los modos (a no ser que la haya cambiado con una orden INK), de forma que todas las líneas han sido de color amarillo intenso (vea las figuras 18 y 2Ø si no está usted seguro sobre este particular).

Para dibujar líneas con colores diferentes se debe emplear una generalización de la orden DRAW. Reinicialice el ordenador y escriba:

```
MOVE 100,100
DRAW 300,300,2
```

El Amstrad ha dibujado una recta que une los puntos (1ØØ, 1ØØ) y (3ØØ, 3ØØ) utilizando la pluma número 2, la cual está cargada con tinta número 2Ø (cyan intenso) en modo 1. Escriba:

```
DRAW 400,0,3
```

El ordenador ha dibujado una recta de (3ØØ,3ØØ) a (4ØØ,Ø) con la pluma 3, que esta cargada con tinta número 6 (roja) en modo 1. (Esta línea puede no ser muy visible sobre el fondo azul.)

Estas órdenes se utilizan con la misma facilidad dentro de los programas. Recuerde que un programa que funcione correctamente en un modo, puede no hacerlo en otro, debido a que los colores de las tintas dependen del modo de pantalla. El siguiente programa dibuja un rectángulo en modo 1, con un lado en amarillo, otro en cyan y los dos restantes en rojo:

```
10 MODE 1
20 MOVE 100,100
30 DRAW 400,100
40 DRAW 400,300.3
50 DRAW 100,300,2
60 DRAW 100,100,3
```

Observe que en la línea 3Ø no se especifica PEN, de forma que el ordenador utiliza PEN 1 automáticamente, dibujando una línea amarilla. Ejecute el programa por segunda vez. Quizá le sorprenda ver que la línea amarilla ha desaparecido.

Si no especificamos ningún número de pluma, como ocurre en la línea 3Ø, el ordenador utiliza la pluma número 1 sólo cuando se trata de la *primera* orden de dibujar que obedece. En cualquier otro caso, utilizará la pluma correspondiente a la *última* orden de dibujar que haya ejecutado. Cuando termina la primera ejecución del programa, la última pluma que se ha utilizado es la 3. En la segunda ejecución, al llegar a la línea 3Ø la recta se dibuja con la pluma 3, ya que la instrucción DRAW de esa línea no especifica otra cosa.

La ventaja que esto ofrece consiste en que cuando se ha establecido un número de pluma en una orden de dibujo, todas las líneas que se trazan a continuación aparecen en el color de esa pluma mientras no se especifique otra distinta:

```
10 MODE 1
20 MOVE 200,100
30 DRAW 400,200,2
40 DRAW 100,350
50 DRAW 200,100
```

Ejecute este programa también en modo 2, en el cual la pluma 2 está cargada con un color diferente.

El modo Ø es el más apropiado para dibujar en colores, siempre que no se exija demasiada resolución:

```
10 REM dibuja una bandera en 10 colores diferent
   es
20 MODE 0
30 PAPER 1
40 CLS
50 REM coordenadas del asta
60 astaizquierda=100
70 astaderecha=120
80 astaabajo=50
90 astaarriba=350
100 topeizquierda=80
110 topederecha=140
120 topearriba=370
130 REM dibujar el asta
```

```
140 MOVE astaizquierda,astaabajo
150 DRAW astaizquierda,astaarriba,0
160 DRAW topeizquierda,topearriba
170 DRAW topederecha,topearriba
180 DRAW astaderecha,astaarriba
190 DRAW astaderecha,astaabajo
200 REM coordenadas de la bandera
210 banderecha=500
220 banarriba=240
230 MOVE astaderecha,banarriba
240 DRAW banderecha,banarriba,2
250 DRAW banderecha,banarriba-100,2
260 MOVE banderecha,banarriba
270 banarriba=banarriba-10
280 DRAW astaderecha,banarriba,3
290 banarriba=banarriba-10
300 DRAW banderecha,banarriba,4
310 banarriba=banarriba-10
320 DRAW astaderecha,banarriba,5
330 banarriba=banarriba-10
340 DRAW banderecha,banarriba,6
350 banarriba=banarriba-10
360 DRAW astaderecha,banarriba,7
370 banarriba=banarriba-10
380 DRAW banderecha,banarriba,8
390 banarriba=banarriba-10
400 DRAW astaderecha,banarriba,9
410 banarriba=banarriba-10
420 DRAW banderecha,banarriba,10
430 banarriba=banarriba-10
440 DRAW astaderecha,banarriba,11
450 banarriba=banarriba-10
460 DRAW banderecha,banarriba,12
470 DRAW astaderecha,banarriba,2
480 REM devolver el fondo a normal
490 PAPER 0
```

Puede bajar la bandera a media asta sin más que modificar la línea 220.

Ejercicios

1. Dibuje una puerta de color verde sobre fondo blanco. Escriba sobre la puerta el número 12 en caracteres azules y dibuje también un buzón en negro, todo ello en modo 0.

2. Modifique el programa anterior para que represente la misma escena, con colores diferentes, en modo 1. (Tendrá que utilizar plumas diferentes, pues puede ocurrir que las plumas que ha seleccionado para el primer programa estén todas cargadas con el mismo color en modo 1).
3. Volviendo al programa de la casa que vimos al principio de este capítulo, modifíquelo de manera que pueda captar por el teclado los colores con que se quiera dibujar las líneas.

Ventanas de colores

Ya hemos visto en este mismo capítulo lo útil que puede resultar la ventana de texto cuando utilizamos instrucciones INPUT. La instrucción WINDOW tiene otras aplicaciones relacionadas con el tratamiento de los gráficos. Se puede trabajar con varias ventanas simultáneamente:

```
10 MODE 1
20 REM definir ventana #0 arriba a la izquierda
30 WINDOW 1,20,1,12
40 REM definir otra ventana abajo a la derecha
50 WINDOW #1,21,40,13,25
60 PRINT "Esto va a la ventana principal de text
   o"
70 PRINT #1,"y esto va a la otra."
```

La ventana de texto "principal" es aquélla a la que se dirigen todas las órdenes tales como PRINT y CLS. Por ejemplo,

```
?"Hola"
```

escribe en la ventana principal de texto. Pero en cambio

```
?#1,"Hola"
```

hace que el mensaje aparezca en la otra ventana. Cada ventana tiene asociado un número; el de la ventana principal es el #Ø. En la línea 5Ø del programa hemos definido la ventana número 1 mediante la instrucción WINDOW #1. La línea 7Ø escribe en esta ventana gracias a la instrucción PRINT #1. Para escribir en la ventana principal de texto se puede especificar PRINT #1, o sencillamente PRINT. Pruebe lo siguiente:

```
?#Ø,"Esta es la ventana #Ø"
?"Esta tambien es la ventana #Ø"
```

La cláusula '#Ø' es opcional cuando trabajamos con la ventana principal de texto. Si no mencionamos ningún número de ventana, el ordenador entiende siempre que nos estamos refiriendo a la ventana #Ø, o sea, a la ventana principal.

Si en algún momento llega a sentirse perdido con esto de las ventanas, vuelva a la situación normal ejecutando una orden MODE.

Lo más interesante de las ventanas es que no sólo se puede escribir en ellas, sino que también se les puede aplicar diferentes colores de PAPER y PEN. Añada las siguientes líneas al programa:

```
51 REM escribir en la pantalla principal
52 PAPER 3
54 CLS
56 PEN 1
60 PRINT "Esto va a la ventana principal de text
   o"
61 REM ahora escribe en la segunda ventana
62 PAPER #1,2
64 CLS #1
66 PEN #1,0
```

En la ventana #Ø, esto es, en la ventana principal, se escribe ahora en caracteres amarillos sobre fondo rojo, mientras que en la #1 se escribe texto azul sobre fondo cyan. Al seleccionar los números de PEN y PAPER para una ventana, es necesario especificar el número de ventana. Ésta es la razón por la que las instrucciones de las líneas comprendidas entre la 62 y la 7Ø contienen '#1', como por ejemplo PEN #1,Ø en la línea 66. De esta forma seleccionamos PEN Ø, que contiene tinta azul, para escribir *sólo* en la ventana #1. Repetimos que, tratándose de la ventana principal, el '#Ø' es *opcional* en lo relativo a las instrucciones PEN y PAPER. También podíamos haber escrito las líneas 52 a 56 de la siguiente forma:

```
52 PAPER #0,3
54 CLS #0
56 PEN #0,1
```

obteniendo exactamente el mismo resultado al ejecutar el programa. Observe que todas las órdenes de manejo de texto que ya conocemos pueden ser aplicadas a ventanas distintas de la principal. Escriba:

```
CLS#1
```

y verá cómo la ventana número 1 se vuelve de color cyan.

Observe que, aunque la instrucción CLS va dirigida a la ventana número 1, los mensajes tales como "Ready" siguen apareciendo en la ventana principal, o sea, en la número Ø. Si lo desea, puede enviar el listado del programa a una ventana distinta de la #Ø:

```
LIST#1
```

Escriba:

```
PEN#1,3
```

Aunque no parece que haya ocurrido nada, hemos cambiado la pluma de la ventana #1. Ahora está preparada para escribir con la pluma número 3, lo que se hará evidente cuando le enviemos algún texto:

```
PRINT#1,"Texto rojo en #1"
```

Podemos cambiar el color de fondo con:

```
PAPER#1,1
```

Tampoco ahora parece que haya ocurrido nada, mientras no escribamos en la ventana:

```
PRINT#1,"Fondo amarillo"
```

Con la orden

```
CLS#1
```

cambiamos el fondo de toda la ventana #1, que pasa a ser ahora de un color amarillo intenso.

Ejercicios

1. Defina dos ventanas de texto, una en la parte superior de la pantalla y otra en la inferior, y escriba su nombre en una y su apellido en la otra.
2. Defina dos ventanas de texto en modo Ø y el mensaje 'WINDOW #Ø' en verde sobre fondo blanco en la ventana principal, y después 'WINDOW #1' en negro sobre fondo rojo en la otra ventana.

Gráficos y ventanas

Podemos hacer incluso que las ventanas se solapen (es decir, que se superpongan parcialmente):

```
10 MODE 1
20 REM definir la ventana principal
30 WINDOW 1,20,8,16
40 PAPER 3
50 CLS
60 REM segunda ventana que se solapa con la prin
   cipal
70 WINDOW #1,11,30,10,18
80 PAPER #1,2
90 CLS #1
```

Observe que la ventana #1 está delante de la ventana #Ø. Cuando las ventanas se solapan, la que se borra en último lugar queda en primer plano; de las restantes se borra la parte que tengan en común con ésta. Si añade al programa la línea

```
100 CLS
```

y borra la línea 5Ø, verá que la ventana #Ø predomina, por ser la última borrada, sobre la #1. En este aspecto, no hay más predominio entre ventanas que el impuesto por el orden en que se las borra y por la superficie que tengan en común.

Por si el solapado de ventanas no fuese bastante complicado, el Amstrad nos permite definir y manejar hasta 8 ventanas de texto distintas, numeradas del Ø al 7:

```
10 MODE 0
20 WINDOW 1,20,23,25
30 INPUT "Dar cuatro colores de pluma (0-16)";pe
   n1,pen2,pen3,pen4
40 INPUT "Ahora cuatro coloresde fondo";paper1,p
   aper2,paper3,paper4
50 REM definir las ventanas y asignarles papel y
    pluma
60 WINDOW #1,6,15,13,22
70 CLS #1
80 PEN #1,pen1
90 PAPER #1,paper1
100 CLS #1
110 REM llevar el cursor al centro de la ventana
120 LOCATE #1,5,5
130 PRINT #1,"1"
140 WINDOW #2,1,8,7,16
150 PEN #2,pen2
160 PAPER #2,paper2
170 CLS #2
180 LOCATE #2,4,5
190 PRINT #2,"2"
200 WINDOW #3,7,14,1,10
210 PEN #3,pen3
220 PAPER #3,paper3
230 CLS #3
240 LOCATE #3,4,5
250 PRINT #3,"3"
260 WINDOW #4,13,20,6,15
270 PEN #4,pen4
280 PAPER #4,paper4
290 CLS #4
300 LOCATE #4,4,5
310 PRINT #4,"4"
```

Este programa demuestra la utilidad que puede llegar a tener la instrucción WINDOW, al proporcionarnos una forma tan fácil y rápida de rellenar con colores zonas rectangulares.

En la línea 120 se utiliza por primera vez la orden LOCATE referida a una ventana. Las coordenadas de texto que se usan con esta instrucción LOCATE están basadas en la nueva ventana. Los ángulos superiores izquierdos de las ventanas tienen coordenadas (1,1); cada rectángulo tiene una anchura de 7 caracteres y una profundidad de 9, de modo que el centro de cada ventana se encuentra en (4,5).

Este programa nos demuestra una vez más las ventajas que ofrecen las variables, y además nos recuerda que cuando tengamos que utilizar números, podemos hacer el programa más flexible asignándolos a variables.

El programa de gráficos que vimos antes no es el óptimo, ya que no disponemos de medios para rellenar zonas de color, aunque sí podemos trazar líneas de diversos colores. Ahora podemos combinar las instrucciones MOVE, DRAW y WINDOW para producir representaciones gráficas muy vistosas. (Recuerde que la pantalla de gráficos nunca se ve afectada por las ventanas de texto y que aquélla ocupa siempre toda la pantalla. Las coordenadas gráficas no dependen, por supuesto, de las ventanas que hayamos definido.)

```
10 MODE 0
11 REM crear techo rojo
20 WINDOW 3,18,1,5
30 PAPER 3
40 CLS
41 REM crear fachada marron
50 WINDOW #1,3,18,6,20
60 PAPER #1,9
70 CLS #1
71 REM crear puerta verde
80 WINDOW #2,10,12,15,20
90 PAPER #2,12
100 CLS #2
101 REM crear ventana amarilla
110 WINDOW #3,6,8,9,12
120 PAPER #3,6
130 CLS #3
131 REM crear segunda ventana azul
140 WINDOW #4,14,17,9,12
150 PAPER #4,6
160 CLS #4
161 REM dibujar vidrios de la ventana
170 MOVE 208,208
180 DRAW 208,272,1
190 MOVE 160,240
200 DRAW 256,240
201 REM dibujar vidrios de la otra ventana
```

```
210 MOVE 416,240
220 DRAW 544,240
230 MOVE 480,208
240 DRAW 480,272
241 REM adornar la puerta
250 MOVE 296,88
260 DRAW 296,168,5
270 DRAW 372,168
280 DRAW 372,88
290 DRAW 296,88
```

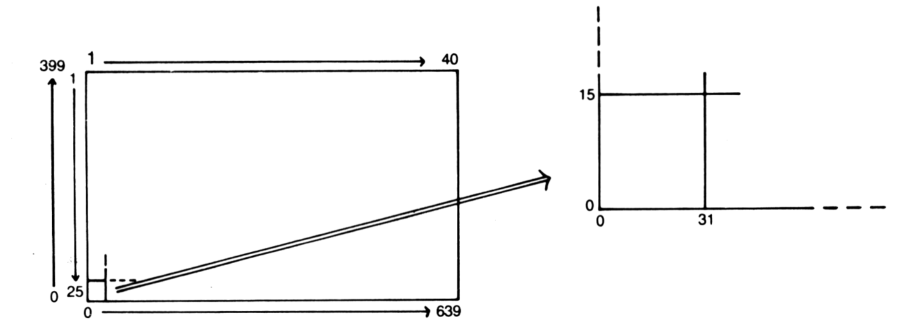


Figura 21. En modo Ø, el espacio dedicado a cada caracter es de 32 puntos de ancho por 16 de alto. El diagrama muestra la relación entre coordenadas de texto y coordenadas gráficas.

Al combinar ventanas con instrucciones de dibujo nos encontramos con el pequeño problema que plantea la utilización de diversos sistemas de coordenadas. Para dibujar con precisión dentro de una ventana, tenemos que calcular en qué coordenadas gráficas empieza la ventana de texto. Si superponemos las coordenadas gráficas y de texto en modo Ø, está claro que el espacio ocupado por cada carácter tiene 32 puntos de ancho y 16 de alto (Fig. 21), de lo que se deduce que, para el modo Ø, las fórmulas que dan las coordenadas del extremo inferior izquierdo de cualquier caracter:

graficosx=(textox−1)*32
graficosy=(25−textoy)*16

Por ejemplo, la puerta verde del programa anterior se ha formado mediante una instrucción WINDOW en la línea 8Ø; el carácter del extremo inferior izquierdo de la ventana tiene las coordenadas (1Ø, 2Ø). De esto se deduce que las coordenadas gráficas del punto extremo inferior izquierdo del interior de la puerta es:

graficosx=(1Ø−1)*32=9*32=288
graficosy=(25−2Ø)*16=5*16=8Ø

o sea, (288, 8Ø). Sabiendo que la esquina inferior izquierda de la puerta está en (28Ø, 8Ø) podemos decidir ya qué coordenadas necesitan las órdenes MOVE y DRAW. Por ejemplo, el dibujo del contorno interior de la puerta comienza en la línea 25Ø con una instrucción MOVE que lleva el cursor gráfico al punto (296, 88). Se escogió este punto porque sabemos que (296, 88) se encuentra algo más arriba y hacia la derecha del ángulo inferior, (28Ø, 8Ø), de la puerta.

Si esto le parece complicado, no se preocupe demasiado. Las conversiones de coordenadas de texto a coordenadas gráficas sólo se necesitan cuando deseamos utilizar ambos tipos de coordenadas para una representación especial en pantalla. Más adelante veremos que estas conversiones son necesarias para la preparación de diagramas o gráficos mezclados con texto, pero también veremos que hay sistemas más fáciles que el arriba descrito.

Ejercicios

1. Defina cuatro ventanas de diferentes colores de tal forma que se obtenga el siguiente diseño:

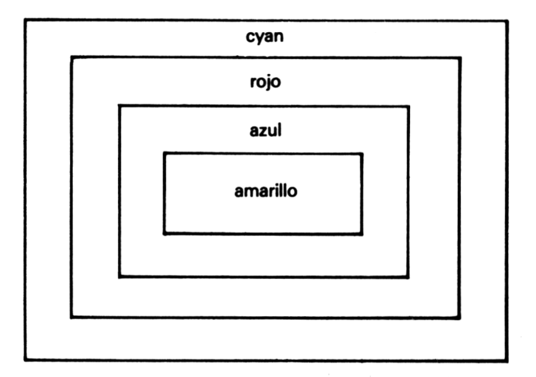


Figura 22. Ventanas de colores.

2. Dibuje en modo Ø un coche rojo con ventanas azules, "ruedas" cuadradas negras y tapacubos blancos.
Intente añadir algunos otros detalles utilizando órdenes MOVE y DRAW.

3. Modifique el programa anterior de modo que el color del coche y la posición en que se lo deba dibujar sean datos que se capten por el teclado.

4. Programe el dibujo de un hombre asomado a la ventana de una casa y saludando con la mano. Utilice las instrucciones WINDOW, MOVE y DRAW.

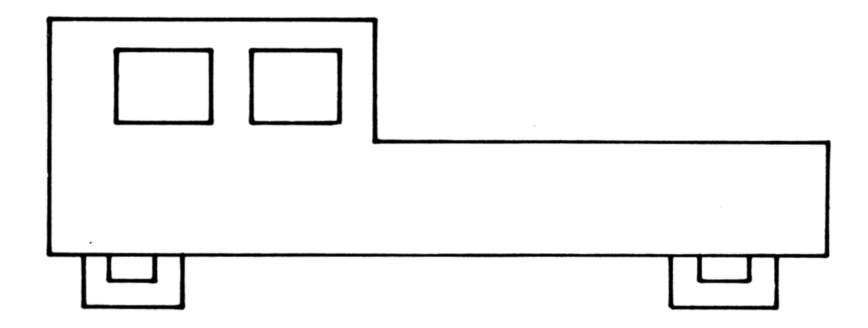


Figura 23. Coche para el ejercicio 2.

4

Bucles

REPETICIÓN DE TAREAS

Todos los programas que hemos visto hasta ahora tenían una cosa en común: al ejecutar el programa, el ordenador obedecía las instrucciones por orden ascendente de números de línea, empezando por el más bajo y terminando por el más alto. No obstante, en algunos casos esto puede tener desventajas, pues puede haber instrucciones que nos gustaría que el ordenador repitiese. Cualquier programa que escribamos contendrá grupos de instrucciones que se ejecutarán linealmente, pero también es probable que contenga grupos de instrucciones cuya ejecución deba repetirse cierto número de veces. Un programa que nos ofrezca la posibilidad de repetir instrucciones será siempre mucho más útil, como demuestra el siguiente:

```
10 MODE 1
20 FOR cuenta=1 TO 100
30 PRINT cuenta
40 NEXT
```

La línea 20 establece un *bucle* de instrucciones. La variable 'cuenta' se utiliza para llevar la cuenta del número de veces que se ha repetido el bucle. En este caso, la 'cuenta' comienza por 1. En la línea 30 se escribe el valor de la variable, que es 1 en la primera ejecución del bucle.

Cada vez que el programa llega a la línea 40, la instrucción NEXT suma 1 a 'cuenta'. A continuación se inicia la siguiente pasada por el bucle, en la que se vuelven a ejecutar todas las instrucciones comprendidas entre FOR ... TO y NEXT, a condición de que 'cuenta' no haya alcanzado el valor límite, 100, establecido en la línea 20. 'Cuenta' es lo que se denomina *variable de control* o también *variable del contador* del bucle.

Todas las instrucciones contenidas entre el FOR de la línea 20 y el NEXT de la 40, serán obedecidas 100 veces. Inserte otra instrucción entre los dos extremos del bucle:

```
10 MODE 1
20 FOR cuenta=1 TO 100
30 PRINT cuenta
35 PRINT "Hola"
40 NEXT
```

En cada pasada por el bucle, el ordenador escribirá el valor de 'cuenta' al llegar a la línea 30, y la palabra "Hola" de la línea 40. Si se modifica la línea 20 de la siguiente forma:

```
10 MODE 1
20 FOR cuenta=5 TO 9
30 PRINT cuenta
35 PRINT "Hola"
40 NEXT
```

verá que el bucle cuenta de 5 a 9. El ordenador ejecuta las líneas 30 y 40 cinco veces en total antes de dar por terminado el bucle. Para mejor fijar esta idea, modifique la línea 20 para que la variable del contador empiece por 3 y termine cuando su valor llegue a 15.

Veamos un pequeño programa que nos da una idea de lo que se puede llegar a hacer con los bucles:

```
10 MODE 1
20 PRINT
30 INPUT "Tabla de multiplicar del ",tabla
40 PRINT
50 FOR cuenta=1 TO 10
60 PRINT cuenta;"por";tabla;"es";cuenta*tabla
70 NEXT
```

Este programa escribe la tabla de multiplicar de cualquier número que introduzcamos en respuesta a la línea 20. La variable de control se puede utilizar siempre en instrucciones internas al bucle, generalmente teniendo cuidado de no modificarla:

```
10 MODE 2
20 FOR coordenadax=1 TO 60
30 PRINT TAB(coordenadax);"TAB en un bucle"
40 NEXT
```

(Lo llamativo en este programa es que, al llenarse la pantalla, la primera línea desaparece y las restantes suben para dejar hueco para una nueva.) La variable de control se puede utilizar también en instrucciones LOCATE:

```
10 MODE 1
20 FOR x=10 TO 30
30 LOCATE x,8
40 PRINT "*";
41 LOCATE x,17
42 PRINT "*";
50 NEXT
60 FOR y=9 TO 16
```

```
70 LOCATE 10,y
80 PRINT "*";SPC(19);"*"
90 NEXT
```

Veamos un par de ejemplos más interesantes:

```
10 MODE 1
20 LOCATE 20,1
30 PRINT "*";
40 FOR y=2 TO 10
50 LOCATE 21-y,y
60 PRINT "*";
70 LOCATE 19+y,y
80 PRINT "*";
90 NEXT
100 LOCATE 10,11
110 FOR cuenta=1 TO 21
120 PRINT "*";
130 NEXT
```

```
10 MODE 0
20 FOR color=0 TO 15
30 PEN color
40 PRINT "Amstrad"
50 NEXT
60 REM vuelta a pluma normal
70 PEN 1
```

En este último programa, la línea 30 cambia de pluma en cada pasada por el bucle, a medida que se va incrementando la variable 'color'. Los bucles producen efectos asombrosos en los programas gráficos. El siguiente dibuja una serie de triángulos concéntricos:

```
10 MODE 1
20 izqx=100
30 fondoy=0
40 derx=500
50 medx=300
60 arribay=300
70 FOR cuenta=1 TO 31
80 MOVE izqx,fondoy
90 DRAW derx,fondoy
100 DRAW medx,arribay
110 DRAW izqx,fondoy
120 izqx=izqx+10
130 fondoy=fondoy+10
```

```
140 derx=derx-10
150 arribay=arribay-10
160 NEXT
```

Los bucles facilitan la programación de totalizadores:

```
10 MODE 1
20 totalgastos=0
30 FOR mes=1 TO 12
40 PRINT "Cuanto ha gastado en el mes";mes;
50 INPUT gastos
60 totalgastos=totalgastos+gastos
70 NEXT
80 PRINT "Pues ha gastado";totalgastos;"pesetas
   en total"
```

Ejercicios

1. Diseñe un programa que escriba su nombre repetidamente hasta llenar la pantalla en modo 1.
2. Diseñe un programa que escriba una M grande formada por asteriscos.
3. Amplíe el programa de "totalgastos" para que se ocupe también de llevar cuenta de sus ahorros mensuales y calcule el total de ahorros anuales.
4. Escriba un programa que dibuje una serie de pentágonos concéntricos. Los colores de los pentágonos y la "velocidad de decrecimiento" del tamaño de los pentágonos deben ser captados por el teclado al comienzo del programa.
5. Diseñe un programa que escriba la palabra "Amstrad" en amarillo sobre los 16 colores de PAPER disponibles en modo Ø.

Cómo controlar los bucles

Los valores extremos de la variable del contador de un bucle pueden a su vez ser variables que el programa puede controlar. Con un bucle podemos definir cómodamente una serie de ventanas:

```
10 MODE 0
11 WINDOW 1,20,20,25
20 INPUT "Cuantas ventanas";ventanas
30 FOR cuenta=1 TO ventanas
40 WINDOW #cuenta,cuenta,cuenta+5,cuenta,cuenta+
   4
50 PAPER #cuenta,cuenta
60 CLS #cuenta
70 NEXT
```

En la línea 30 el límite superior del bucle es una variable. En la línea 40 se definen las ventanas en posiciones que dependen del valor de la variable del contador. Trate de modificar este programa para que defina ventanas de tamaño variable. He aquí otro ejemplo en el que el límite superior del bucle capta por el teclado:

```
10 MODE 1
20 INPUT "Cuantas lineas quiere ";lineas
30 FOR cuenta=1 TO lineas
40 MOVE cuenta*10,0
50 DRAW 320,350
60 NEXT
```

En respuesta a la línea 20 se puede introducir cualquier número, pero un valor demasiado alto hará que el programa tarde lo suyo en terminar. En la línea 50 el segundo extremo de las rectas es fijo, pero también podríamos hacerlo aleatorio, o incluso dependiente de la variable del contador:

```
10 MODE 1
20 INPUT "Cuantas lineas quiere ";lineas
30 FOR cuenta=1 TO lineas
40 MOVE cuenta*10,0
50 DRAW 500,cuenta*10
60 NEXT
```

Cuando se trabaja con coordenadas gráficas, no nos interesa que la variable de control del bucle avance de 1 en 1, porque eso es demasiado lento. En el programa anterior hubo que multiplicar la variable por 10 para conseguir que los extremos de las rectas estuviesen suficientemente separados unos de otros. En efecto, compruebe qué ocurre si sustituye 'cuenta*10' por 'cuenta' en las líneas 40 y 50.

Podemos controlar la tasa de variación de la variable del contador incluyendo en la definición del bucle la cláusula STEP (paso):

```
10 REM dibujar 9 rectangulos
20 MODE 0
30 FOR cuenta=0 TO 500 STEP 60
40 MOVE cuenta, 0
50 DRAW cuenta+50,0
60 DRAW cuenta+50,200
70 DRAW cuenta,200
80 DRAW cuenta,0
90 NEXT
```

En la línea 30 damos a 'cuenta' un valor incial de 0; al llegar a la línea 90, este valor no aumenta de 1 en 1, sino de 60 en 60, que es el valor del paso. Observe que el extremo superior del bucle no tiene por qué ser un múltiplo del valor del paso. En este ejemplo, 'cuenta' recorre los valores 0, 60, 120, 180, 240, 300, 360, 420 y 480.

Al llegar a 54Ø, el bucle ya no se repite porque este valor es mayor que 5ØØ. El valor del paso también puede ser una variable. Incluya en el programa anterior una instrucción INPUT que capte el valor del paso. Pruebe con diferentes valores y observe los resultados.

El siguiente programa es similar, pero más interesante. Dibuja una serie de cuadrados que pueden solaparse. Pruébelo, por ejemplo, con los valores 5 y 5Ø.

Ejercicios

1. Escriba un programa que dibuje una malla rectángular. (Para ello serán necesarios dos bucles: uno que dibuje las rectas verticales y otro que dibuje las horizontales.)
2. Escriba un programa que acepte como entrada las ventas de varios meses y dibuje (en modo Ø) un gráfico de barras en el que se representen las ventas de cada mes en un color diferente. (Si dibuja las columnas con órdenes DRAW, sólo podrá trazar los perfiles; si las dibuja con ventanas de texto, podrá rellenarlas, pero la precisión será bastante menor.)
3. Nuestro viejo amigo, el propietario del restaurante, ya ha oído hablar de los bucles y ahora se le ha antojado un programa que acepte como entrada el número de artículos que ha vendido y el precio de cada uno y que calcule el importe total, que calcule el 12 por ciento de IVA y que lo sume para escribir la cantidad que debe pagar el cliente. Escríbale usted el programa.
4. Escriba un programa que dibuje un triángulo formado por asteriscos en cualquier lugar de la pantalla en el color que se introduzca por el teclado en respuesta a una instrucción INPUT.

Control con STEP

El valor del paso que se pone tras STEP en los bucles FOR ... NEXT puede ser negativo o incluso fraccionario. Si utilizamos pasos negativos el valor de partida del bucle FOR ... NEXT ha de ser mayor que el de parada:

```
10 MODE 1
20 FOR cuentaatras=10 TO 1 STEP-1
30 PRINT cuentaatras
40 NEXT
50 PRINT "IGNICION!!!!!!!!!!!!!!!!"
```

Los pasos con decimales se utilizan con frecuencia en los programas de gráficos en que los figuras que se dibujan no son simples triángulos ni rectángulos, sino polígonos más complicados. Las coordenadas de los vértices se calculan fácilmente utilizando senos y cosenos, cuyos valores son menores que 1.

El siguiente programa nos permite trazar cualquier figura poligonal, desde un triángulo hasta una pseudo-circunferencia. El usuario introduce por el teclado la posición del centro de la figura, el radio y el número de lados:

```
10 MODE 1
20 INPUT "Radio de la figura";radio
30 INPUT "Coordenadas X e Y del centro";centrox,
   centroy
40 INPUT "Numero de lados";lados
50 CLS
60 paso=2*PI/lados
70 MOVE centrox,centroy+radio
80 FOR angulo=0 TO 2*PI STEP paso
90 DRAW centrox+radio*SIN(angulo),centroy+radio*
   COS(angulo)
100 NEXT
110 DRAW centrox,centroy+radio
```

Ejercicios

1. Escriba un programa de IGNICION!!! en modo Ø, en el que los números de la cuenta atrás aparezcan dentro de columnas de tamaño decreciente, proporcional al valor de la variable del contador.
2. Modifique el programa del polígono para que dibuje dos polígonos concéntricos de colores diferentes.

Datos dentro del programa

Supongamos que queremos escribir un programa que tome los nombres y las notas de un grupo de alumnos, realice con ellas los cálculos necesarios y escriba las notas finales. Podría tratarse de un grupo de 1ØØ alumnos, por lo que no parece adecuado tener que suministrar al ordenador por el teclado los nombres y las notas cada vez que ejecutemos el programa. Tampoco es aconsejable utilizar una variable para cada dato cuando éstos son muy numerosos. BASIC da una solución a este problema: los datos se pueden guardar en líneas de programa, precedidos de la palabra clave DATA, para luego ser leídos por una instrucción READ:

```
10 MODE 1
20 FOR alumno=1 TO 5
30 READ nombrealumno$,nota1,nota2
40 nota1porciento=nota1/60*100
50 nota2porciento=nota2/80*100
60 promedio=(nota1porciento+nota2porciento)/2
70 PRINT nombrealumno$;nota1porciento;nota2porci
   ento;promedio
```

```
80 NEXT
90 DATA Garcia,40,20,Sanchez,10,16
100 DATA Ramos
110 DATA 7,34,Rojo,55
120 DATA 76,Lopez,58,71
```

La salida de este programa es bastante confusa. Más tarde la arreglaremos, ahora vamos a estudiar el funcionamiento de las instrucciones que acabamos de presentar. La instrucción READ de la línea 30 hace que el ordenador examine el programa hasta que encuentre una línea encabezada con la palabra 'DATA'. La primera de estas líneas es la 90. A continuación lee el primer elemento de la lista de datos, que es 'García', y lo asigna a la primera variable de la línea 30, 'nombrealumno$'.

Pero en la instrucción READ figuran otras dos variables, de modo que el ordenador tiene que seguir leyendo datos. El número 40 lo asigna a la variable nota1, y el 20 a nota2.

Como nota 1 es una puntuación sobre un total de 60 puntos, la línea 40 la convierte a su equivalente sobre 100 puntos. Análogamente actúa la línea 50. La nota media se calcula en la línea 60, y en la 70 se escriben los datos y los resultados de los cálculos.

La línea 80 es una instrucción NEXT que hace que el programa vuelva al principio del bucle, línea 20, tras incrementar la variable del contador. En esta segunda pasada por el bucle la línea 30 actúa de forma diferente: cuando el ordenador ejecuta por segunda vez una instrucción READ, ya sabe dónde encontrar los datos; de hecho, lleva la cuenta de cuántos datos ha leído para saber cuál es el que tiene que leer a continuación. Así, en la segunda ejecución de la línea 30 lee 'Sánchez' y asigna esta cadena literal a la variable 'nombrealumno$'.

El proceso se repite hasta que se han leído todos los nombres y notas y el bucle termina. Nótese que, cada vez que se agotan los datos de una línea, el ordenador busca la siguiente línea DATA y continúa leyendo en ella. Los datos deben estar separados por comas. El programa funciona igual si los datos se colocan de forma más organizada:

```
90 DATA Garcia,40,40
95 DATA Sanchez,10,16
100 DATA Ramos,7,34
110 DATA Rojo,55,76
120 DATA Lopez,58,71
```

Para detectar y corregir errores es preferible que las líneas DATA no sean demasiado largas. Por otra parte, las líneas DATA pueden estar en *cualquier* lugar del programa, no necesariamente antes de las instrucciones READ que leen los datos. El ordenador ignora las líneas DATA hasta el momento de ejecutar una instrucción READ. Así pues, las líneas DATA pueden estar al principio del programa, aunque es más frecuente ponerlas al final; lo que no es recomendable es desperdigarlas por todo el programa.

En ocasiones un mismo grupo de datos ha de ser utilizado varias veces por el pro-

grama. Por ejemplo, en un programa como el anterior, podríamos necesitar escribir los nombres y las notas en dos fases del programa. No es necesario duplicar las líneas DATA, ya que al ordenador le podemos decir dónde debe empezar a leer los datos mediante la orden RESTORE. Si añadimos la línea

```
25 RESTORE 95
```

al programa anterior, veremos que sólo lee el nombre y las notas de 'Sánchez'. La instrucción RESTORE 95 obliga a iniciar la lectura de datos en la línea 95; cada vez que se repite el bucle, el ordenador lee los mismos datos que en la pasada anterior. Borre la línea 25 e introduzca la siguiente:

```
15 RESTORE 100
```

Ahora, al entrar en el bucle, el ordenador sabe que tiene que leer los datos empezando por la línea 1ØØ. En la segunda pasada lee los de la 11Ø; en la tercera, los de la 12Ø. Pero en la cuarta pasada ya no queda ninguna línea posterior por leer y el ordenador emite un mensaje de error.

La instrucción RESTORE a secas, sin número de línea, hace que el "puntero de datos" se sitúe señalando la primera línea DATA del programa.

El siguiente programa utiliza la instrucción RESTORE para dibujar una serie de monigotes en diferentes lugares de la pantalla:

```
10 MODE 1
20 FOR cuenta=100 TO 500 STEP 100
30 RESTORE
40 READ piernax,piernay,inglex,ingley,piernax1,p
   iernay1
50 READ brazox,brazoy,mediox,medioy,brazox1,braz
   oy1
60 READ cabezax,cabezay,oidox,oidoy,oidox1,oidoy
   1
70 MOVE piernax+cuenta,piernay:DRAW inglex+cuent
   a,ingley
80 MOVE piernax1+cuenta,piernay1:DRAW inglex+cue
   nta,ingley
90 DRAW mediox+cuenta,medioy:DRAW brazox+cuenta,
   brazoy
100 MOVE brazox1+cuenta,brazoy1:DRAW mediox+cuen
    ta,medioy
110 DRAW cabezax+cuenta,cabezay:DRAW oidox+cuent
    a,oidoy:DRAW oidox1+cuenta,oidoy:DRAW cabeza
    x+cuenta,cabezay
120 NEXT
130 DATA 35,105,70,150,90,110
140 DATA 60,170,80,180,95,160
150 DATA 80,190,65,205,85,210
```

Organización de los cálculos aritméticos

Volviendo al programa de cálculo de las notas medias, podemos mejorarlo en diversos sentidos. En primer lugar, es evidente que tenemos que arreglárnoslas para redondear las notas. El BASIC del Amstrad dispone de varias instrucciones de redondeo. Una de ellas es INT, que redondea "hacia abajo", esto es, al primer número entero que es menor que el dado:

```
40 nota1porciento=INT(nota1/60*100)
50 nota2porciento=INT(nota2/80*100)
60 promedio=INT((nota1porciento+nota2porciento)/
   2)
```

Observe el número que queremos redondear lo escribimos entre corchetes a la derecha de INT. Este método de redondeo es injusto para los alumnos porque, en caso de que la nota media no sea un número entero, siempre salen perdiendo. Es preferible, pues, utilizar con la instrucción CINT, que da el número entero *más próximo* al que se desea redondear. De esta forma, 3.4 se redondea hacia abajo, mientras que 3.5 se redondea hacia arriba:

```
40 nota1porciento=CINT(nota1/60*100)
50 nota2porciento=CINT(nota2/80*100)
60 promedio=CINT((nota1porciento+nota2porciento)
   /2)
```

Ejercicios

1. Mejore el programa de las notas medias organizando mejor la salida, esto es, ordenando la información en columnas y poniendo una cabecera en cada columna.
2. Modifique el programa de representación gráfica de las cifras de ventas de forma que lea los datos en líneas DATA antes de dibujar el gráfico de barras.
3. Escriba un programa que lea en líneas DATA las coordenadas de texto y los números de PAPER de siete ventanas antes de definir las ventanas.
4. Escriba un programa que lea las coordenadas gráficas y los números de PEN de una serie de rectas y que luego las trace para formar el dibujo de un barco.

NÚMEROS ALEATORIOS

Aprovechando que ya conocemos las funciones INT y CINT, es hora de presentar una función que es extraordinariamente útil en los programas de juegos. Se trata de RND, que genera números aleatorios. Escriba:

```
?rnd
```

El ordenador ha escrito un número, elegido al azar, igual o mayor que Ø y menor que 1. Si vuelve a escribir lo mismo obtendrá un número completamente diferente.

Pero los números aleatorios menores que 1 no tienen demasiado interés en los juegos. Por ejemplo, si tratamos de simular el lanzamiento de una pareja de dados, necesitamos números aleatorios que sean enteros y puedan oscilar entre Ø y 12, así que tendremos que transformar los números generados por RND para que sean del tipo y margen deseados. El método general es el siguiente:

1) Generar el número aleatorio (con RND).
2) Restar el extremo inferior del margen del extremo superior y sumar 1.
3) Multiplicar el número así obtenido por el número aleatorio.
4) Sumar a este el resultado el extremo inferior del margen.
5) Redondear el nuevo resultado con INT.

Por si el método le parece confuso, aquí tiene un programa que se lo aclara:

```
10 MODE 1
20 INPUT "Minimo numero aleatorio";menor
30 INPUT "Maximo numero aleatorio";mayor
31 REM Escribir 20 numeros aleatorios entre los
   dos
40 FOR cuenta=1 TO 20
50 numaleat=INT(RND*(mayor-menor+1)+menor)
60 PRINT numaleat
70 NEXT
```

Como decíamos, los números aleatorios son la base de muchos juegos. El siguiente programa simula el lanzamiento de dos dados:

```
10 MODE 1
20 INPUT "Cuantas veces echo los dos dados";tira
   das
30 FOR cuenta=1 TO tiradas
40 dado1=INT(RND*6+1)
50 dado2=INT(RND*6+1)
60 totaldados=dado1+dado2
70 PRINT "En la tirada";cuenta;"salieron";totald
   ados;"dados"
80 NEXT
```

También resulta divertido introducirlos en programas gráficos. El siguiente programa dibuja un número aleatorio de triángulos en posiciones elegidas aleatoriamente:

```
10 MODE 0
20 numdetriang=INT(RND*100+1)
```

```
30 FOR cuenta=1 TO numdetriang
40 izqx=INT(RND*640)
50 derx=INT(RND*640)
60 medx=INT(RND*640)
70 izqy=INT(RND*400)
80 dery=INT(RND*400)
90 medy=INT(RND*400)
100 color=INT(RND*16)
110 MOVE izqx,izqy
120 DRAW derx,dery,color
130 DRAW medx,medy
140 DRAW izqx,izqy
150 NEXT
```

Observe en este programa que, dado que el extremo inferior del margen de las coordenadas gráficas es Ø, no hay nada que sumar en la etapa 4) del método de obtención de números aleatorios enteros. Por eso las expresiones que generan las coordenadas gráficas, líneas 4Ø a 9Ø, son más sencillas.

Ejercicios

1. Diseñe un programa que escriba en la pantalla una serie de números aleatorios enteros comprendidos entre 1 y 1Ø.
2. Modifique el programa de los triángulos aleatorios para que dibuje también rectángulos o algún otro polígono. Puede incluir ventanas, pero no olvide que los márgenes de las coordenadas de texto son distintos de los de las coordenadas gráficas.
3. Utilice números aleatorios como parámetros de LOCATE para escribir 1ØØ asteriscos de colores aleatorios (en modo Ø).

Matemáticas con números aleatorios

Otra aplicación frecuente de los números aleatorios son los problemas de aritmética, en este caso las tablas de multiplicar:

```
10 MODE 1
20 FOR preguntas=1 TO 10
30 numero1=INT(RND*12+1)
40 numero2=INT(RND*12+1)
50 PRINT"Cuanto es";numero1;"por";numero2;
60 INPUT respuesta
70 NEXT
```

Desde luego, sería preferible que el ordenador pudiese decir si la respuesta es correcta; de no serlo, prodríamos repetir la pregunta para dar otra oportunidad antes de pasar a la pregunta siguiente. Eso de "repetir" nos hace pensar en un bucle, ¿verdad? Lo que ocurre es que el tipo de bucle que conocemos no sirve en esta situación. El bucle FOR ... NEXT siempre termina como resultado de una *operación de contar*. El ordenador cuenta el número de veces que se ha ejecutado el bucle y lo interrumpe cuando ese número ha alcanzado cierto valor preestablecido.

Sin embargo, hay ocasiones en las que no sabemos de antemano cuántas veces debe ser ejecutado el bucle. El ejemplo anterior, en el que se debe repetir una pregunta hasta que la respuesta sea correcta, es una de ellas; necesitamos un bucle, pero no del tipo FOR ... NEXT.

EL BUCLE WHILE ... WEND

El bucle de tipo WHILE ... WEND tiene la solución para nuestro problema. Lo que lo distingue del bucle FOR ... NEXT es que el bucle WHILE ... WEND termina cuando se cumple cierta *condición*, **no** cuando se lo ha ejecutado cierto número de veces. Su mecanismo es el siguiente:

WHILE (mientras que) la respuesta sea incorrecta
formular otra vez la pregunta
wend (fin de *while*)

Veamos un pequeño programa que hace una pregunta y la repite hasta que la respuesta sea correcta:

```
10 MODE 1
20 respuesta=0
30 numero1=INT(RND*12+1)
40 numero2=INT(RND*12+1)
50 WHILE respuesta<>numero1*numero2
60 PRINT "Cuanto es";numero1;"por";numero2;
70 INPUT respuesta
80 WEND
```

En la línea 50 se incluye un signo < >, que puede ser desconocido para el lector. Significa "distinto de". Así, para la línea 50 se podría leer de la siguiente forma: "mientras la respuesta sea distinta del producto de numero 1 por numero 2, repetir las instrucciones siguientes". El final del grupo de instrucciones que forman el bucle está señalado por la palabra clave WEND (contracción de WHILE y END), línea 80. Obsérvese la diferencia con respecto a FOR ... NEXT. Aquí no tenemos idea de cuándo terminará el bucle, pero sí sabemos qué tiene que ocurrir para que termine.

El bucle WHILE ... WEND se programa cuando se desea restringir la entrada por el teclado a un margen de valores determinado. Uno de los problemas de la pro-

gramación es que del usuario hay que esperar siempre lo peor. Si un programa pide al usuario que teclee un número menor que 1Ø, es casi seguro que introducirá un número mayor "para ver qué pasa". Si el programa está bien diseñado, examinará la respuesta y, si ésta está fuera del margen permitido, la rechazará y solicitará otra. Esto es muy fácil de programar con un bucle WHILE:

```
10 MODE 1
20 respuesta=11
30 WHILE respuesta>10
40 INPUT "Por favor, un numero menor que 10: ",r
   espuesta
50 WEND
```

Este programa contiene algunas novedades. En la línea 3Ø aparece el signo >, que significa "mayor que". Así pues, la línea se lee: "mientras la respuesta sea mayor que 1Ø, repetir las instrucciones siguientes". Al ejecutar el programa, el ordenador solicita el número incansablemente, hasta que el usuario decide portarse bien, momento en el que el bucle termina.

En la línea 2Ø se hace la variable 'respuesta' igual a 11. Es *muy importante*, cuando se va a entrar en un bucle WHILE, asegurarse de que en el punto de entrada la condición no se cumple, de modo que el bucle se ejecute al menos una vez. Si no fuera por la línea 2Ø, al ejecutar el programa no se observaría ningún efecto. Esto es debido a que BASIC considera que todas las variables valen Ø, a menos que se les haya asignado otro valor; si no se hubiera declarado ningún valor para 'respuesta', al llegar a la línea 3Ø la condición se cumpliría, ya que Ø es menor que 1Ø, y el bucle no llegaría a ejecutarse ni siquiera una vez.

WHILE permite realizar con facilidad bucles infinitos que el usuario puede interrumpir cuando desee sin más que introducir un valor apropiado. Por ejemplo, con un bucle FOR . . . NEXT podemos construir un programa que sume nuestros gastos mensuales:

```
10 MODE 1
20 totalgastos=0
30 INPUT "Cuantos objetos ha comprado";objetos
40 FOR cuenta=1 TO objetos
50 PRINT "Cuanto ha costado el objeto";cuenta;
60 INPUT gasto
70 totalgastos=totalgastos+gasto
80 NEXT
90 PRINT "El gasto total fue de";totalgastos;"pe
   setas"
```

Pero si lo diseñamos con WHILE ya no necesitamos decirle al ordenador cuántos datos vamos a suministrarle:

```
10 MODE 1
20 totalgastos=0
```

```
30 respuesta$="S"
40 WHILE respuesta$="S"
50 INPUT "Importe del gasto: ",gasto
60 totalgastos=totalgastos+gasto
70 INPUT "Algo mas (S/N)";respuesta$
80 WEND
90 PRINT "El gasto total fue de";totalgastos;"pe
   setas"
```

Si estamos leyendo datos almacenados en líneas DATA, podemos incluir un *terminador de datos* para que el bucle sepa cuándo debe dejar de leerlos:

```
10 MODE 1
20 READ nombre$,numtelef$
30 WHILE nombre$<>"XXX"
40 PRINT nombre$,numtelef$
50 READ nombre$,numtelef$
60 WEND
70 DATA Alberto,923 234557,Isabel,911 221339,Jai
   me,245 4633,Diana,241 4358
80 DATA Rolando,733 3234,Pili,865 2148,Pepe,933
   433 833,XXX,YYY
```

El terminador de datos es un valor que ponemos al final de los datos; tiene que ser radicalmente distinto de los datos normales para que no quepa la posibilidad de que el programa interprete como terminador lo que en realidad es un dato. En este caso el terminador es 'XXX'. La línea 30 lee nombres y números de teléfono hasta que el nombre leído sea XXX. Lo que ocurre es que, como la línea 50 lee los datos de dos en dos, después de XXX tenemos que incluir *algo* para evitar el error 'DATA exhausted' (''datos agotados'') que de otra forma se produciría. Aquí hemos puesto 'YYY', pero habría valido cualquier otra cosa.

Volviendo a la cuestión de la elección de terminadores, hay que evitar a toda costa que el programa los confunda con los datos. Por ejemplo, en un programa de proceso de notas de examen, no sería aconsejable utilizar el cero como terminador; alguien podría habérselas ingeniado para sacar un cero, y entonces la lectura de los datos terminaría prematuramente. En un caso como éste, un terminador adecuado sería, por ejemplo, −99.

Algunos aspectos del último programa pueden haberle llamado la atención. ¿Por qué hay una operación de lectura fuera del bucle (línea 20)? Esto nos resuelve dos problemas. En primer lugar, puede ocurrir que en las líneas DATA no haya más datos que el terminador. La línea 20 garantiza entonces que el bucle no llegue a ejecutarse, pues la condición de terminación del bucle se cumple en el mismo momento en que se intenta iniciarlo.

El segundo problema es el siguiente: si no fuera por la línea 20, el terminador aparecería en la lista de salida al final de los nombres. Compruébelo con el siguiente programa, más sencillo y a primera vista tan bueno como el anterior:

```
10 MODE 1
30 WHILE nombre$<>"XXX"
40 READ nombre$,numtelef$
50 PRINT nombre$,numtelef$
60 WEND
70 DATA Alberto,923 234557,Isabel,911 221339,Jai
   me,245 4633,Diana,241 4358
80 DATA Rolando,733 3234,Pili,865 2148,Pepe,933
   433 833,XXX,YYY
```

Esto evidencia un detalle que hay que cuidar en la programación de bucles WHILE: si se utiliza como condición para terminar el bucle, el hecho de que una variable tome un determinado valor, hay que cerciorarse de que la variable tome ese valor *al final* del bucle. En este último ejemplo, leíamos la variable nombre$ al principio del bucle y por eso la línea 50 puede escribir 'XXX'. En cambio, en el anterior, la lectura la realizábamos al final.

Ejercicios

1. Escriba un programa que lea en listas de datos los nombres, edades y fechas de nacimiento de sus amigos y los escriba ordenadamente en la pantalla. Utilice un terminador adecuado.
2. Escriba un programa que permita dibujar directamente con el teclado. El programa dibujará rectas desde el último punto visitado hasta un punto cuyas coordenadas debe suministrar el usuario. Defina una ventana de texto a través de la cual el usuario pueda introducir los sucesivos pares de coordenadas X, Y. (Por este procedimiento no será posible "levantar la pluma" al dibujar.) Utilice un bucle WHILE como núcleo del programa; haga que el bucle termine cuando el usuario responda con 'N' a la pregunta '¿Más coordenadas?'.

AND y OR

A veces la condición de terminación del bucle WHILE ... WEND no puede ser sencilla, sino combinación de varias. En un programa en el que hacemos preguntas y esperamos una respuesta del usuario, podemos querer que la pregunta se repita indefinidamente hasta que la respuesta sea correcta. Pero parece más lógico repetirla solamente un pequeño número de veces, el número de oportunidades que queramos dar al usuario. La forma de especificar el bucle podría ser: "WHILE (mientras) la repuesta sea incorrecta AND (y además) el número de intentos sea igual o menor que 3 ... ". Veamos un ejemplo:

```
10 intentos=0
20 respuesta=0
```

```
3Ø primnum=INT(RND*12+1)
4Ø segnum=INT(RND*12+1)
5Ø WHILE respuesta<>primnum*segnum AND intentos<
   3
6Ø PRINT "Cuanto es";primnum;"por";segnum;
7Ø INPUT respuesta
8Ø intentos=intentos+1
9Ø WEND
```

En el bucle hemos incluído un contador de intentos, líneas 1Ø, 5Ø y 8Ø. La línea 5Ø especifica en qué condiciones se puede ejecutar el bucle. En este caso, se tiene que cumplir que la respuesta sea incorrecta y que además el número de intentos sea menor que 3. Para que el ordenador pueda ejecutar las instrucciones del bucle es necesario que se cumplan *las dos* condiciones. Ejecute el programa varias veces y compruebe que el bucle se interrumpe solamente cuando se da la respuesta correcta o cuando se han dado tres respuestas incorrectas.

Las líneas 1Ø, 5Ø y 8Ø, que son las que tienen que ver con el contador de intentos, son muy importantes. Es muy fácil confundirse al preparar un contador; con frecuencia ocurre que termina demasiado tarde, o demasiado pronto, o que no termina nunca. Aquí el contador parte de Ø, línea 1Ø; cada intento se contabiliza *después* de producirse, línea 8Ø. Después del tercer intento, la variable 'intentos' es igual a 3 y el bucle ya no puede repetirse porque entonces ya no se cumple una de las condiciones de la línea 5Ø.

Para fijar las ideas, observe qué ocurre si en la línea 1Ø se hace el número de intentos igual a 1, lo que no deja de ser un número razonable ya que se va a hacer el primer intento. Verá que entonces tiene que cambiar las condiciones de la línea 5Ø.

El siguiente programa es un juego en el que se dan 8 oportunidades para acertar un número aleatorio entero comprendido entre 1 y 1ØØ.

```
1Ø MODE 1
2Ø intentos=1
3Ø numaleat=INT(RND*1ØØ+1)
4Ø adiv=Ø
5Ø WHILE adiv<>numaleat AND intentos<9
6Ø PEN 1
7Ø PRINT
8Ø PRINT"Oportunidad";
9Ø PEN 3
1ØØ PRINT intentos;:PEN 1:PRINT"para adivinar un
     numero"
11Ø PEN 2
12Ø PRINT
13Ø INPUT "Escriba un numero:";adiv
14Ø IF adiv<numaleat THEN PRINT"demasiado bajo"
15Ø IF adiv>numaleat THEN PRINT"demasiado alto"
16Ø intentos=intentos+1
17Ø WEND
```

Las líneas 14Ø y 15Ø dan un anticipo de lo que vamos a ver en el siguiente capítulo. Hemos incluído un poco de color para hacer más atractivo el programa. En esta ocasión el número de intentos se iguala a 1, no a Ø, en la línea 2Ø. No es que queramos complicar las cosas; si no comprende por qué lo hemos hecho, ponga un Ø en la línea 2Ø y observe qué ocurre. Dicho sea de paso, y aunque parezca mentira, el número se puede adivinar *siempre* en 8 intentos o menos, a condición de que se siga el método adecuado.

En la definición de bucles WHILE, las condiciones se pueden combinar también con OR ("o"). Por ejemplo, en un juego de "tres en raya" podría haber un par de líneas como las siguientes:

```
500 WHILE gana=0 OR movimientos<9
510 PRINT"Cual es su movimiento";
```

donde la variable 'gana' se pone a Ø al principio del juego y a 1 cuando gana uno de los dos jugadores. La estructura del bucle sería:

WHILE (mientras) ninguno de los dos haya ganado
OR (o)
haya huecos libres en el tablero,
seguir jugando
WEND

Esto representa que el juego continúa si se cumple *cualquiera* de las dos condiciones, y que termina solamente en el caso de que no se cumpla *ninguna* de ellas.

También podemos utilizar OR en un bucle WHILE para limitar las respuestas introducidas por el teclado a un margen de valores permisibles. En un ejemplo anterior sólo vigilábamos que el número introducido fuera menor que 1Ø. Ahora podemos comprobar que está *entre* dos valores extremos:

```
10 MODE 1
20 INPUT "Escriba un numero del 1 al 10: ",nume
   ro
30 WHILE numero<5 OR numero>10
40 PRINT "No es ese."
50 INPUT "Escriba otro: ",numero
60 WEND
70 PRINT "El";numero;"es correcto!"
```

La línea 3Ø inicia el bucle, cuya acción principal es pedir un número al usuario. El bucle se repite siempre que se cumpla cualquiera de las dos condiciones. Hay muchos juegos, semejantes al de "tres en raya", "barcos", etc. en los que es necesario restringir los movimientos para que no se salgan del tablero. Como hemos visto, el bucle WHILE puede servir para rechazar los movimientos incorrectos.

La comprobación de los datos de entrada (generalmente por el teclado) es lo que se denomina *validación de datos*. Se trata de un mecanismo que debe estar presente en casi todos los programas, tanto en los de juegos como en los educativos y comerciales.

Ejercicios

1. Escriba un programa que capte por el teclado una coordenada X y rechace los valores que queden fuera de la pantalla. (Los valores permisibles irán, pues, del Ø al 639, ambos inclusive.)
2. Escriba un programa que escriba una tira de asteriscos en cualquier lugar de la pantalla. Debe captar por teclado las coordenadas X e Y de texto del lugar donde debe empezar la escritura y la longitud de la tira; debe rechazar, lógicamente, las coordenadas que sean inválidas y las longitudes que, de ser aceptadas, hicieran que la tira no cupiese en el resto de la línea. (Serán necesarios tres bucles WHILE, uno para cada dato.)

5

Acciones condicionadas

TOMA DE DECISIONES

En el capítulo anterior empezamos a desarrollar un sencillo programa que hacía preguntas y comprobaba la validez de las respuestas; pero teníamos la limitación de que no sabíamos cómo tomar decisiones en función de la evolución de los acontecimientos. Por ejemplo, nos gustaría que el ordenador diera la respuesta correcta cuando el usuario no es capaz de aceptarla. Para ello necesitamos saber cómo decirle al ordenador:

1) Si la respuesta es incorrecta, da tú la correcta.
2) En cambio, si es correcta, formula la siguiente pregunta.

Podemos elegir entre dos cursos de acción si utilizamos la instrucción IF ... THEN (si ... entonces ...):

```
10 MODE 1
20 INPUT"Cuantos años tienes";edad
30 IF edad>18 THEN PRINT"Ya no eres un adolescen
   te."
```

Ejecute este programa varias veces introduciendo edades diferentes. Puede comprobar que el mensaje de la línea 30 aparece solamente cuando la edad introducida es mayor que 18. El ordenador examina la afirmación comprendida entre IF y THEN; si su valor lógico resulta ser "verdadero", ejecuta el resto de la línea. Si su valor es "falso" (en este caso, si la edad es igual o menor que 18), ignora el resto de la línea, esto es, no ejecuta las instrucciones que haya después de THEN, sino que pasa a la siguiente línea del programa.

Podemos ampliar el programa incluyendo otras instrucciones IF ... THEN que controlen la acción para otros márgenes de edad:

```
10 MODE 1
20 INPUT"Cuantos años tienes";edad
30 IF edad>18 THEN PRINT"Ya no eres un adolescen
   te."
40 IF edad<=14 THEN PRINT"Todavia estas en E.G.B
   ."
```

```
50 IF edad>=65 THEN PRINT"Me parece que ya estas
   retirado."
60 IF edad>100 THEN PRINT"Mas de cien! Felicidad
   es!"
```

La línea 40 comprueba si la edad es igual o menor que 14; la línea 50, si es igual o mayor que 65. Observe que si introducimos el número 67, se cumplen dos de las condiciones, y por lo tanto obtenemos dos mensajes. Si introducimos 110 obtenemos tres mensajes.

Las instrucciones que pongamos detrás de THEN no tienen que ser necesariamente PRINT, sino que pueden ser cualquier orden válida de BASIC. Por ejemplo, en nuestro programa de preguntas sobre la tabla de multiplicar, podemos utilizar IF ... THEN para llevar la cuenta del número de respuestas incorrectas:

```
10 MODE 1
20 fallos=0
30 FOR preguntas=1 TO 10
40 num1=INT(RND*12+1)
50 num2=INT(RND*12+1)
60 PRINT "Cuanto es";num1;"por";num2;
70 INPUT respuesta
80 IF respuesta<>num1*num2 THEN fallos=fallos+1
90 NEXT
100 PRINT
110 PRINT "Has tenido";fallos;"fallos"
```

La línea 80 comprueba si la respuesta es correcta. Si es incorrecta, suma 1 a la variable "fallos".

El programa sería de mayor calidad si cada vez que el usuario da una respuesta incorrecta, el ordenador pudiera decirle cuál es la correcta. Podríamos incluir otras dos instrucciones IF ... THEN:

```
80 IF respuesta<>num1*num2 THEN fallos=fallos+1
85 IF respuesta<>num1*num2 THEN PRINT "No, eso n
   o es correcto."
86 IF respuesta<>num1*num2 THEN PRINT "La respue
   sta es";num1*num2
```

Pero esto parece tedioso y reiterativo. Una vez comprobada la respuesta en la línea 80, ¿por qué tenemos que comprobarla de nuevo? En realidad, después de THEN podemos poner varias instrucciones en la misma línea:

```
80 IF respuesta<>num1*num2 THEN fallos=fallos+1:
   PRINT "No, eso no es correcto.":PRINT "La res
   puesta es";num1*num2
```

Si la respuesta es incorrecta, el ordenador ejecuta *todas* las instrucciones que siguen al THEN de la línea 80. Las instrucciones tienen que estar separadas por signos de punto y coma y deben estar todas en la misma línea.

Esta forma de agrupar instrucciones en una línea es válida en cualquier lugar del programa, no sólo a la derecha de una sentencia IF ... THEN:

```
10 MODE 1:fallos=0:FOR preguntas=1 TO 10:num1=IN
   T(RND*12+1):num2=INT(RND*12+1):PRINT"Cuanto e
   s";num1;"por";num2;:INPUT respuesta
20 IF respuesta<>num1*num2 THEN fallos=fallos+1:
   PRINT"No, eso no es correcto.":PRINT "La resp
   uesta es";num1*num2
30 NEXT:PRINT:PRINT"Has tenido";fallos;"fallos."
```

Ahora bien, el lector no debe pensar que esto es una panacea. En realidad, los programas tan "compactos" son difíciles de leer y editar. A no ser que el programa sea muy largo y ocupe gran cantidad de memoria, es preferible dejar las líneas más bien cortas, con dos o tres instrucciones como máximo en cada una, salvo, naturalmente, en el caso de las que contengan instrucciones IF ... THEN.

El programa anterior puede servir como base para todos los programas que funcionen a base de preguntas y respuestas. Con ligeras modificaciones, se lo puede adaptar para que haga preguntas sobre geografía, francés, etc. Veamos un ejemplo:

```
10 MODE 1
20 fallos=0
30 FOR preguntas=1 TO 12
40 READ mes$,dias
50 PRINT "Cuantos dias tiene ";mes$;
60 INPUT respuesta
70 IF respuesta<>dias THEN fallos=fallos+1:PRINT
   "No, eso no es correcto; ";mes$;" tiene";dias
   ;"dias"
80 NEXT
90 PRINT
100 PRINT "Has tenido";fallos;"fallos"
110 DATA enero,31,febrero,28,marzo,31,abril,30,m
    ayo,31,junio,30
120 DATA julio,31,agosto,31,setiembre,30,octubre
    ,31,noviembre,30,diciembre,31
```

Los datos para las preguntas y las respuestas son los nombres de los meses y los días que tiene cada uno de ellos. En un programa más general, es posible que los datos deban contener las preguntas completas; las líneas DATA serían entonces de la forma:

```
200 DATA Cual es la capital del Reino Unido,Lond
    res
210 DATA Que rio pasa por Valladolid,el Duero
```

Ejercicios

1. Escriba un programa que acepte como entrada por el teclado la estatura y el peso del usuario y haga unos comentarios en función de los valores introducidos.
2. Escriba un programa que lea una lista de nombres y puntuaciones y que escriba los nombres poniendo 'APROBADO' al lado de los que tengan 45 o más puntos, y 'SUSPENSO' al lado de los restantes. Utilice dos colores diferentes para estas dos palabras.
3. Utilice la instrucción IF ... THEN para mejorar el programa que hace preguntas sobre la tabla de multiplicar en el siguiente sentido: cuando se han formulado 10 preguntas, el ordenador debe emitir diferentes mensajes en función del número de respuestas correctas. Utilice un número de PEN distinto para cada mensaje.
4. Diseñe su propio programa basado en preguntas y respuestas. Incluya un contador de las respuestas correctas. Haga que al final del programa se escriba un mensaje que dependa del número de respuestas correctas y convierta ese número en un porcentaje sobre el total de preguntas.

IF ... THEN ... ELSE

En el programa de preguntas sobre la tabla de multiplicar podemos incluir una línea que escriba el mensaje adecuado cada vez que el usuario dé una respuesta correcta:

```
10 MODE 1
20 fallos=0
30 FOR preguntas=1 TO 10
40 num1=INT(RND*12+1)
50 num2=INT(RND*12+1)
60 PRINT "Cuánto es";num1;"por";num2;
70 INPUT respuesta
80 IF respuesta<>num1*num2 THEN fallos=fallos+1:
   PRINT"No, eso no es correcto.":PRINT"La respu
   esta es";num1*num2
85 IF respuesta=num1*num2 THEN PRINT"Correcto. M
   uy bien!"
90 NEXT
100 PRINT
110 PRINT "Has tenido";fallos;"fallos."
```

Nos encontramos aquí con una situación muy frecuente en informática: una respuesta sólo puede ser correcta o incorrecta, sin ambigüedad ni posibilidades intermedias. Al comprobar la validez de una respuesta, sólo hay dos resultados posibles, y podemos utilizar una ampliación de la instrucción IF .. THEN para controlar los dos cursos de acción correspondientes:

```
80 IF respuesta<>num1*num2 THEN fallos=fallos+1:
   PRINT"No, eso no es correcto.":PRINT"La respu
   esta es";num1*num2 ELSE PRINT"Correcto. Muy B
   ien."
```

La línea 8Ø viene a decir: "IF (si) la respuesta es incorrecta THEN (entonces) haz tal cosa, ELSE (en caso contrario) haz tal otra". Si la afirmación que se comprueba fuera más compleja, los resultados posibles serían más de dos y entonces se podrían enlazar varias sentencias IF ... THEN ... ELSE para tomar las decisiones oportunas.

Veamos un ejemplo en el que se utiliza IF ... THEN ... ELSE para dibujar un diagrama de barras de las temperaturas medias mensuales:

```
10 MODE 0
11 REM preparar ejes
20 ycero=100
30 MOVE 35,0
40 DRAW 35,399
50 MOVE 35,ycero
60 DRAW 600,ycero
61 REM dibujar diagrama de barras
70 FOR mes=1 TO 12
80 READ temperatura
90 IF temperatura<0 THEN color=6 ELSE color=3
100 temperaturaescala=temperatura*12
110 MOVE mes*40,ycero
120 DRAW mes*40,ycero+temperaturaescala,color
130 DRAW mes*40+35,ycero+temperaturaescala
140 DRAW mes*40+35,ycero
150 NEXT
160 DATA -5,-1,0,5,11,15,20,22,14,12,10,-3
```

En realidad, con la temperatura puede ocurrir una de estas tres cosas: o es mayor que Ø, o es igual a Ø o es menor que Ø. Si no queremos incluir la temperatura Ø en ninguno de los otros dos grupos, necesitamos otra línea:

```
95 IF temperatura=0 THEN color=12
```

El siguiente programa, basado en IF ... THEN ... ELSE, demuestra hasta qué punto los ordenadores pueden parecer inteligentes cuando en realidad no están haciendo prácticamente nada:

```
10 MODE 1
20 INPUT "Como te llamas";nombre$
30 PRINT:PRINT "Encantado de conocerte, ";nombre
   $
40 PRINT:PRINT"Estoy pensando en un animal y qui
   ero queintentes adivinar su nombre."
50 PRINT:PRINT"Puedes hacerme las preguntas que
   quierasacerca del aspecto del animal, pero lu
   ego solo tendras una oportunidad para acertar
   su nombre."
60 PRINT:PRINT"Cuando creas que lo sabes, respon
   de   S cuando te pregunte si quieres hacer tu
   unico intento."
70 PRINT:PRINT"Bueno, cual es tu primera pregunt
   a?"
80 respuesta$="n"
90 preguntas=0
100 WHILE respuesta$<>"S"
110 PRINT
120 preguntas=preguntas+1
130 LINE INPUT"",pregunta$
140 IF RND(1)>0.5 THEN PRINT"Si" ELSE PRINT"No"
150 PRINT:INPUT "Estas ya preparado intentarlo (
    S/N)";respuesta$
160 IF respuesta$<>"S" THEN PRINT:PRINT"Cual es
    tu siguiente pregunta?"
170 WEND
180 PRINT:INPUT"Cual crees que es el animal";ani
    mal$
190 PRINT
200 IF RND(1)>0.5 THEN PRINT"Si!!! Lo adivinaste
     con";preguntas;"preguntas." ELSE PRINT"No,
    lo siento, te has equivocado. Y has hecho";p
    reguntas;"preguntas!"
```

Ejercicios

1. Modifique el programa que califica alumnos con 'SUSPENSO' y 'APROBADO' de forma que la selección se realice con IF ... THEN ... ELSE.
2. Escriba un programa que lea en líneas DATA una serie de nombres y edades y dé una relación de los nombres en dos columnas: una para los que tienen derecho a voto y otras para los que son demasiado jóvenes para votar.
3. Escriba un programa que permita introducir po el teclado datos sobre ingresos y gastos, estos últimos en forma de números negativos. Disponga una ventana de texto para escribir en ella una columna con los ingresos y otra con los gastos, con los respectivos totales al final. Escriba los gastos en rojo y los ingresos en otro color.

Entrada por el teclado: otro método

Cuando un programa ofrece al usuario múltiples opciones, a menudo es cómodo utilizar una serie de sentencias IF . . . THEN para controlarlas. El siguiente programa convierte la pantalla en un tablero de dibujo controlado por el teclado:

```
10 MODE 1
20 direccion$=""
30 coordx=320
40 coordy=200
50 WINDOW 1,40,24,25
60 WHILE direccion$<>"x"
70 INPUT "Que direccion (i/d/a/b)";direccion$
80 IF direccion$="b" THEN coordy=coordy-1
90 IF direccion$="a" THEN coordy=coordy+1
100 IF direccion$="i" THEN coordx=coordx-1
110 IF direccion$="d" THEN coordx=coordx+1
120 PLOT coordx,coordy,1
130 WEND
```

El programa se detiene introduciendo 'x'.

Hay algunos detalles dignos de observación en este programa. En primer lugar, puesto que las posibilidades de movimiento de la pluma son cuatro, no tenemos más remedio que utilizar varias sentencias IF . . . THEN. En segundo lugar, el programa no funciona con el teclado en situación de CAPS LOCK, ya que las sentencias IF . . . THEN, tal como las hemos puesto, sólo reconocen letras minúsculas. Finalmente, la necesidad de pulsar ⟨ENTER⟩ después de cada letra es irritante. ¿No habrá otra forma más cómoda de utilizar el teclado?

La hay. En lugar de INPUT utilizamos INKEY$:

```
10 MODE 1
20 direccion$=""
30 coordx=320
40 coordy=200
60 WHILE direccion$<>"x"
70 direccion$=INKEY$
80 IF direccion$="b" THEN coordy=coordy-1
90 IF direccion$="a" THEN coordy=coordy+1
100 IF direccion$="i" THEN coordx=coordx-1
110 IF direccion$="d" THEN coordx=coordx+1
120 PLOT coordx,coordy,1
130 WEND
```

El programa funciona ahora de forma completamente distinta. INKEY$ examina continuamente la situación del teclado y reconoce el estado de las teclas sin necesidad de pulsar ⟨ENTER⟩. Para dibujar una recta hacia la izquierda basta con pulsar

la tecla 'i'. Como la tecla es de repetición, al mantenerla pulsada, INKEY$ recoge una sucesión de letras 'i'. Estas letras se transfieren, una a una, a la variable 'direccion$', línea 7Ø. Es esta variable la que es objeto de comprobación en las líneas 8Ø a 11Ø, en las que se decide en qué dirección se debe mover la pluma o si se debe detener el programa.

INKEY$ es la función que se debe utilizar siempre que la respuesta del usuario a través del teclado pueda consistir en un solo carácter. Observe que INKEY$ no *espera* hasta que se produzca una pulsación en el teclado, sino que está examinándolo continuamente, aunque no haya ninguna tecla pulsada. Compruébelo con este programa:

```
10 MODE 1
20 respuesta$="n"
30 WHILE respuesta$<>"s"
40 PRINT "Estas ya listo para parar (s/n)?"
50 respuesta$=INKEY$
60 WEND
```

El ordenador escribe repetidamente el mensaje porque INKEY$ explora el teclado, no detecta respuesta y la línea 3Ø permite que el programa continúe. El bucle se repite hasta que el usuario reacciona y pulsa la 's'.

INKEY$ permite controlar el tiempo que tarda el usario en dar la respuesta. Con ella podemos hacer intervenir el factor tiempo en los programas de juegos. Ésta es la base del programa de familiarización con el teclado que viene en la cinta de demostración del ordenador. Todaviía no sabemos cómo generar letras al azar, así que el siguiente programa es una versión rudimentaria en el que sólo se ofrece al usuario una letra, leída en una línea DATA:

```
10 MODE 1
15 PEN 1
20 PRINT "Voy a escribir una letra en la"
30 PRINT "pantalla y quiero que la encuentres lo
   "
40 PRINT "mas deprisa posible en el teclado y qu
   e"
50 PRINT "la pulses."
60 PEN 3
70 PRINT:PRINT"Seras cronometrado!"
80 PRINT:PRINT "Pulsa R cuando estes preparado"
90 respuesta$=INKEY$
100 WHILE respuesta$<>"R" AND respuesta$<>"r"
110 respuesta$=INKEY$
120 WEND
130 arranque=TIME
140 READ letra$
150 PRINT:PRINT "La letra es ";
```

```
160 PEN 2
170 PRINT letra$
180 respuesta$=INKEY$
190 WHILE respuesta$=""
200 respuesta$=INKEY$
210 WEND
220 total=TIME-arranque
230 PRINT:PRINT "Has tardado";total/300;"segundo
    s"
240 IF respuesta$=letra$ THEN PRINT"pero al meno
    s acertaste" ELSE PRINT"y ademas te equivoca
    ste!"
250 DATA q
```

Las líneas 90 a 120 examinan continuamente el teclado hasta que el usuario pulsa la tecla de la 'R'. Observe que el bucle WHILE que se establece en la línea 100 se repite mientras la respuesta no sea ni 'R' ni 'r'. O sea, el bucle se interrumpe cuando se pulsa la tecla de la 'R', tanto en mayúsculas como en minúsculas.

En la línea 130 empezamos a cronometrar. TIME es una función intrínseca del BASIC del Amstrad que da, en unidades de 1/300 segundos, el tiempo transcurrido desde que se encendió la máquina. Escriba

```
?TIME
```

y el ordenador le dirá cuánto tiempo lleva encendido. El ordenador actualiza el valor de TIME continuamente, como puede comprobar volviendo a escribir ?TIME. Lo que hacemos en la línea 130 es guardar en la variable 'arranque' el valor de TIME en el instante en que se empieza a cronometrar.

Las líneas 180 a 210 realizan otro examen del teclado, pero esta vez no en busca de una respuesta concreta, sino de la pulsación de *cualquier* tecla. En cuanto se pulsa una tecla, la prueba termina; la línea 220 calcula entonces la diferencia entre el tiempo actual y el inicial. La línea 230 informa del tiempo transcurrido en segundos.

Este programa introduce algunas ideas nuevas, a pesar de que está incompleto y tendremos que ampliarlo más adelante. TIME es una función muy útil de la que podremos servirnos siempre que necesitemos cronometrar algo.

Ejercicios

1. Vuelva al programa de preguntas aleatorias sobre la tabla de multiplicar e incorpore en él algún sistema de cronometrado. Haga que el programa escriba al final de la prueba el tiempo total transcurrido junto con algún comentario sobre si el usuario ha sido lento, normal o rápido.
2. Escriba un programa que escriba repetidamente el valor de TIME convertido a minutos y segundos y que se detenga cuando el usuario pulse la tecla 'P'.

3. Amplíe el programa de dibujo para que permita también dibujar rectas en diagonal con sólo pulsar una tecla. Incluya la posibilidad de dibujar con el color del papel, lo que permitirá borrar líneas dibujadas antes y mover la pluma por la pantalla sin dejar huella. (El problema que se encontrará es que, al elegir esta opción, ya no se sabe en qué lugar de la pantalla está la pluma; tendrá que dibujar dos puntos en cada posición: uno con el color del papel para borrar y otro con un color visible para poder identificar la posición de la pluma.)

Condiciones compuestas en IF ... THEN

Las condiciones que se incluyen en las sentencias IF ... THEN pueden ser expresiones complejas, construídas con las operaciones lógicas AND y OR. Nuestro programa de preguntas sobre la tabla de multiplicar podría contener líneas de este tipo:

```
100 IF tiempototal>1000000 AND respuestas<5 THEN
     PRINT "Ademas de tonto eres lento!"
110 IF tiempototal<5000 AND respuestas<5 THEN PR
    INT "Mas te vale correr menos y acertar mas!"
```

En un programa que seleccione nuevos alumnos para una famosa universidad podría haber una línea como esta:

```
200 IF ingresos>4000000 OR CI>130 THEN PRINT"Bie
    nvenido a la Universidad!"
```

El siguiente programa mueve un punto por la pantalla y lo hace rebotar cuando choca con el borde:

```
10 MODE 0
20 fin=0
30 coordx=300
40 coordy=100
50 cambiox=4
60 cambioy=8
70 WHILE fin<1
80 IF coordx+cambiox<0 OR coordx+cambiox>639 THE
   N cambiox=-cambiox
90 IF coordy+cambioy<0 OR coordy+cambioy>399 THE
   N cambioy=-cambioy
100 coordx=coordx+cambiox
110 coordy=coordy+cambioy
120 color=INT(RND*15+1)
130 PLOT coordx,coordy,color
140 WEND
```

El punto parte de las coordenadas (3ØØ, 1ØØ), líneas 3Ø y 4Ø. Las líneas 5Ø y 6Ø establecen la amplitud de los movimientos. El bucle WHILE es indefinido; para salir de él hay que pulsar ⟨ESC⟩ dos veces. Las líneas 8Ø y 9Ø calculan el valor futuro de las coordenadas gráficas; si alguna de ellas es exterior a la pantalla, las instrucciones IF ... THEN invierten el sentido del movimiento en el eje correspondiente, consiguiéndose así el efecto de rebote. El movimiento se puede confinar a un área más pequeña cambiando los valores Ø, 399 y 639 que aparecen en las líneas 8Ø y 9Ø.

Ejercicios

1. Modifique este último programa para que el punto cambie de color al rebotar.
2. Escriba un programa que lea en líneas DATA los nombres y número de sentencias condenatorias de una serie de delincuentes. Seleccione los que hayan sido condenados 1Ø veces o más y escriba en la pantalla la lista de sus nombres, bajo el encabezamiento de 'REINCIDENTES PELIGROSOS'.
3. El departamento comercial de una empresa clasifica sus clientes como buenos pagadores (1), normales (2) y morosos (3). Escriba un programa que lea una lista de clientes y riesgo actual con ellos y que escriba toda la información disponible sobre los que tengan calificación 3 o deban más de 1ØØØØØ pesetas.

6

Cadenas literales

TODO MENOS NÚMEROS

En los capítulos anteriores hemos concertado nuestra atención en las variables numéricas y en las instrucciones con las que podemos manejarlas. Las funciones que se aplican a las variables literales son también muy potentes. Algunas de las operaciones que realizábamos con variables numéricas son también válidas para las literales, aunque con significado diferente:

```
10 MODE 1
20 INPUT "Cual es tu nombre";nombre$
30 INPUT "Cual es tu apellido";apellido$
40 total$=nombre$+" "+apellido$
50 PRINT "El nombre completo es: ";total$
```

Las cadenas literales se pueden unir (o *concatenar*) sin más que poner un signo '+' entre ellas. También podemos comparar cadenas con los signos =, > y <, tal como hacíamos con los números:

```
10 MODE 1
20 INPUT "Escribe la primera palabra: ",primera$
30 INPUT "Escribe la segunda palabra: ",segunda$
40 IF primera$=segunda$ THEN PRINT "Las dos pala
   bras son identicas.":END
50 IF primera$<segunda$ THEN PRINT primera$;" es
   ta antes que ";segunda$;" en el diccionario."
    ELSE PRINT segunda$;" esta antes que ";prime
   ra$;" en el diccionario."
```

En el caso de las cadenas, la igualdad es la identidad absoluta, carácter por carácter. Por ejemplo, el ordenador considera que 'Manzana' es una cadena distinta de 'manzana'. Los ordenadores tienen sus propias ideas acerca de la ordenación alfabética. De hecho, para la máquina todas las letras mayúsculas van antes que las minúsculas en el orden alfabético.

Como habrá observado al ejecutar el programa anterior, el ordenador interpreta el signo '<', aplicado a cadenas literales, con el significado de 'anterior en orden

alfabético'. La línea 4Ø de este programa termina con una instrucción que aún no conocíamos: END (fin). Su efecto es terminar el programa. El ordenador lo hace sin que se lo pidamos explícitamente cuando se le acaban las líneas, pero en este caso nos interesaba poner END en la línea 4Ø para impedir que la ejecución continuase en la 5Ø. Suprima el END y observe qué ocurre cuando ejecuta el programa e introduce dos cadenas idénticas.

La capacidad de comparar cadenas es fundamental en los trabajos de ordenación. Es muy frecuente tener que ordenar nombres, y la máquina puede realizar este tedioso trabajo para nosotros. Por ejemplo, podemos formar una *base de datos* con información sobre algún tema de nuestro interés; el ordenador puede examinarla rápidamente y extraer de ella la información que nos interese en cada momento. Así, si en un programa tenemos grabada información sobre los nombres y fechas de nacimiento de nuestros amigos, el ordenador puede averiguar con gran rapidez el cumpleaños de un amigo en cuanto le demos el nombre:

```
10 MODE 1
20 INPUT "Como se llama tu amigo";amigo$
30 WHILE nombre$<>"XXX" AND nombre$<>amigo$
40 READ nombre$,cumple$
50 IF nombre$=amigo$ THEN PRINT amigo$;" cumple
   años el ";cumple$
60 WEND
70 IF nombre$="XXX" THEN PRINT "Lo siento. No co
   nozco a tu amigo."
80 DATA JUANITA RODRIGUEZ,29 de octubre,LUIS GOM
   EZ,16 de mayo,MARIANO PEREZ,3 de junio
90 DATA XXX,XXX
```

Por supuesto, cuando hay tan pocos datos, la velocidad a la que trabaja el ordenador no representa gran ventaja. Veremos más adelante que los datos del programa se pueden guardar fuera de él. Cuando hay miles de datos que examinar, la velocidad del ordenador sí es de agradecer.

Un problema que usted puede haber descubierto en este programa tiene que ver con algo que anunciábamos al principio del capítulo. El ordenador no reconoce el nombre de nuestro amigo a menos que lo escribamos *exactamente* tal y como figura en la lista de datos. En algunos ordenadores este problema sólo se resuelve mediante una complicada manipulación de las cadenas introducidas. En el Amstrad todo es más sencillo. Escriba

```
?upper$("Pedro")
```

(En inglés, *upper case* significa "mayúsculas"). UPPER$ convierte todas las letras a mayúsculas. Si le decimos que *lower case* significa "minúsculas", no le será difícil averiguar para qué sirve la función LOWER$:

```
?lower$("Pedro")
```

Podemos mejorar nuestro programa añadiéndole la siguiente línea:

```
25 amigo$=UPPER$(amigo$)
```

Esta línea pide al ordenador que tome la cadena literal que hay almacenada en la variable 'amigo$', la convierta en mayúsculas y luego la vuelva a guardar en la misma variable. Si ejecuta nuevamente el programa, verá que ahora puede escribir los nombres en mayúsculas o en minúsculas. Para que este método funcione es necesario que haya una consistencia en la forma de poner los datos en las líneas DATA. De poco sirve convertir la entrada con UPPER$ o LOWER$ si en los datos hay una mezcla de mayúsculas y minúsculas.

Ejercicios

1. Escriba un programa que acepte como entrada su nombre, su apellido y su título (Sr., Sra., Dr., etc.) y que combine las tres cadenas para escribir el nombre completo.
2. Escriba un programa de léxico inglés-español que escriba la palabra inglesa equivalente a la que se introduce por el teclado.
3. Escriba un programa que acepte como entrada tres palabras y las escriba en orden alfabético. (No es tan fácil como usted cree).

Longitud de una cadena

En muchas ocasiones es necesario poder averiguar la longitud de una cadena. Por ejemplo, si vamos a poner como cabecera de una columna una cadena captada por el teclado, tenemos que estar seguros de que la cadena tecleada no es más larga que la anchura de la columna. O bien, si debemos centrar una frase en la pantalla, necesitamos saber cuál es su longitud para calcular en qué posición debemos empezar a escribirla. La función que determina la longitud de las cadenas es LEN. Escriba

```
?len("Juan")
```

El ordenador ha dado la respuesta correcta, 4. Como ya hemos anunciado, podemos utilizar LEN para centrar frases:

```
10 MODE 1
20 INPUT "Como te llamas";nombre$
30 CLS
40 longitud=LEN(nombre$)
50 posicionx=(40-longitud)/2
60 LOCATE 18,11
70 PEN 3
80 PRINT "hola"
```

```
90 LOCATE posicionx,13
100 PEN 2
110 PRINT nombre$
```

La línea 40 resta la longitud de la cadena de la longitud de línea (que es 40 en modo 1) para determinar cuántos espacios van a quedar libres. Para que la frase quede centrada, los espacios se deben repartir, la mitad a cada lado. Por eso, al dividir por 2 el número de espacios, obtenemos la coordenada de texto X en la que debemos empezar a escribir la cadena. En otros ordenadores habría que redondear este número con INT, pero a la orden LOCATE de la línea 90 no parece importarle el que uno de sus parámetros no sea entero.

Si observa con atención, verá que algunas cadenas no quedan perfectamente centradas. Esto se debe a que cuando la cadena es de longitud impar los espacios sobrantes no se pueden repartir a partes iguales.

Veamos otro ejemplo de aplicación de LEN:

```
10 MODE 1
20 INPUT "De que longitud quieres buscar palabra
   s";longitud
30 READ palabra$
40 WHILE palabra$<>"XXX"
50 IF LEN(palabra$)=longitud THEN PRINT palabra$
60 READ palabra$
70 WEND
80 DATA una,caracteristica,de,la,informatica,se,
   hizo,"patente,",y,todavia,sigue siendo,valida
   ,hoy,"dia:"
90 DATA los,ordenadores,se,estaban,haciendo,cada
   ,vez,mas,"pequeños,",y,sobre,todo,mas,baratos
100 DATA XXX
```

La línea 90 de este programa puede parecerle extraña. La explicación es la siguiente: el ordenador interpreta la coma como separador de datos; la única forma de incluir una coma en un dato es encerrar la cadena entre comillas. Por eso hemos tenido que poner entre comillas "patente," y "pequeños.".

El Amstrad puede repetir un carácter para formar una cadena de la longitud especificada:

```
?string$(20,"A")
```

Si no fuera por esta función, para conseguir el mismo tendríamos que formar un bucle que añadiese en cada pasada un carácter a la cadena.

```
10 MODE 1
20 INPUT "Base del rectangulo: ",base
30 INPUT "Altura: ",altura
```

```
40 CLS
50 LOCATE 1,1
60 PRINT STRING$(base,"*")
70 FOR posiciony=1 TO altura-2
80 PRINT "*";SPC(base-2);"*"
90 NEXT
100 PRINT STRING$(base,"*")
```

En la línea 70 hemos puesto 'altura−2' porque las dos cadenas de asteriscos escritas en las líneas 60 y 100 también cuentan para la altura. Por razón análoga hemos puesto 'base−2' en la línea 80.

Ejercicios

1. Modifique el programa que busca palabras de una longitud especificada en el siguiente sentido: el programa debe escribir el texto completo, llenando en lo posible las líneas de la pantalla, y distinguir las palabras seleccionadas escribiéndolas en un color diferente del de las restantes.
2. Una función típica de los procesadores de texto es la búsqueda y sustitución de palabras, mediante la cual el usuario hace que el ordenador busque en todo el texto una palabra determinada y la cambie por otra. Diseñe un programa que lea un texto en líneas DATA, busque una palabra para sustituirla por otra siempre que aparezca y escriba el texto modificado en la pantalla.
3. Escriba un programa que acepte como entrada su nombre y luego lo escriba centrado dentro de un rectángulo formado por asteriscos.

Eslabones sueltos

El programa que buscaba cadenas de una longitud determinada tenía un pequeño defecto: consideraba que la cadena "dia:" tenía cuatro letras.

Podemos eliminar los signos de puntuación examinando el último carácter de la cadena y suprimiéndolo si no es una letra. Para ello necesitaremos una de las funciones de manejo de cadenas que extraen *subcadenas* de una cadena dada. Veamos un ejemplo:

```
?left$("Hola, que tal",4)
```

(En inglés, *left* significa "izquierda".) Esta orden extrae de la cadena de entrada los cuatro primeros caracteres de la izquierda. Otro ejemplo sería

```
?left$("entrada/salida",7)
```

lo que nos da 'entrada'. Sabiendo que *right* significa derecha, puede imaginar cuál será el efecto de la función RIGHT$:

```
?right$("Barato",4)
```

Efectivamente, extrae cuatro caracteres por la derecha. La tercera función de este grupo es más versátil, pues puede sustituir a LEFT$ y RIGHT$ y además extraer caracteres del interior de las cadenas:

```
?mid$("Instituto",6,2)
```

(*Middle* significa "medio".) En este ejemplo, MID$ genera una subcadena extrayendo 2 caracteres a partir del sexto: 'tu'. Si no se incluye el segundo número, los caracteres que se extraen son todos los restantes, de modo que, por ejemplo,

```
?mid$("Recompensar",6)
```

produce 'pensar'.

Como era de esperar, estas funciones son aplicables también a variables literales. Pero volvamos a nuestro problema original. Tenemos que suprimir el último carácter de la cadena siempre que no sea una letra. Introduzcamos la siguiente modificación en el programa:

```
40 WHILE palabra$<>"XXX"
41 finpalabra$=RIGHT$(palabra$,1)
42 longpalabra=LEN(palabra$)
43 IF finpalabra$="." OR finpalabra$="," OR finp
   alabra$=":" THEN longpalabra=longpalabra-1
50 IF longpalabra=longitud THEN PRINT palabra$
60 READ palabra$
70 WEND
100 DATA XXX
```

La línea 41 toma el último carácter de la cadena. Si es un signo de puntuación, la línea 43 decrementa el valor de la variable 'longpalabra'.

MID$ puede ayudarnos a eliminar los espacios no deseados que pueda haber en las cadenas introducidas por el teclado. Ya hemos visto en un programa anterior que el ordenador distingue "Alvarez" de "ALVAREZ", a pesar de que a los humanos ambas palabras nos puedan parecer la misma. Este problema lo resolvíamos aplicando UPPER$ o LOWER$ a la cadena de entrada. Pero a este respecto hay otra fuente de conflictos: los espacios que el usuario puede teclear antes, después y en medio de las cadenas. El ordenador ignora los espacios que se teclean antes y después de las cadenas, pero no los interiores. Por consiguiente, distingue entre 'E.Alvarez', 'E. Alvarez' y 'E.   Alvarez'; sólo la primera de estas cadenas será reconocida al comparlas con lo almacenado en las líneas DATA. Como solución, se puede adoptar la norma de guardar en las líneas de DATA las cadenas sin espacios y luego suprimir sistemáticamente todos los espacios de las cadenas captadas por el teclado.

Eso es lo que hace el siguiente programa. Usted puede mezclarlo con el programa que pregunta el nombre del amigo y busca la fecha del cumpleaños. Para hacer más patente la diferencia entre la cadena de entrada y la elaborada por el programa, las líneas 120 y 170 escriben las dos cadenas flanqueadas por asteriscos:

```
10 MODE 1
20 INPUT "Escriba una frase: ",frase$
30 longfrase=LEN(frase$)
40 nuevafrase$=""
50 FOR caracter=1 TO longfrase
60 letra$=MID$(frase$,caracter,1)
70 IF letra$<>" " THEN nuevafrase$=nuevafrase$+l
   etra$
80 NEXT
90 PRINT
100 PRINT "La frase original era: "
110 PRINT
120 PRINT STRING$(10,"*");frase$;STRING$(10,"*")
130 PRINT
140 PRINT
150 PRINT "La frase nueva sin espacios es: "
160 PRINT
170 PRINT STRING$(10,"*");nuevafrase$;STRING$(10
   ,"*")
```

También podemos utilizar MID$ para poner en clave un mensaje captado por el teclado:

```
10 MODE 1
20 INPUT "Escribe el mensaje que quieras que pon
   gaen clave: ",palabra$
30 clave$=palabra$
40 longitud=LEN(palabra$)
50 FOR lio=1 TO longitud
60 aleat=INT(RND*longitud+1)
70 der=longitud-aleat
80 clave$=LEFT$(clave$,aleat-1)+RIGHT$(clave$,de
   r)+MID$(clave$,aleat,1)
90 NEXT
100 PEN 1
110 PRINT:PRINT "El mensaje en clave es: ";
120 PEN 3
130 PRINT clave$
140 PRINT
150 REM vuelta a la pluma normal
160 PEN 1
```

El bucle de las líneas 6Ø a 11Ø revuelve las letras de la frase introducida. Podríamos haber hecho que el bucle se repitiese, por ejemplo, 5 veces, pero entonces las frases largas no quedarían suficientemente deformadas, así que hemos preferido que el límite superior del bucle fuera precisamente la longitud de la cadena de entrada. Lo primero que hacemos es elegir en la línea 8Ø una posición de partida. Después, en la línea 9Ø, calculamos cuántas letras quedan a la derecha de esa posición. Finalmente, en la línea 1ØØ juntamos los dos trozos de la cadena, excluyendo la letra que está en la posición seleccionada al azar, y ponemos esa letra al final.

Ejercicios

1. Escriba un programa que capte por el teclado una palabra y la escriba en la pantalla dedicando una línea a cada letra.
2. Modifique el programa del ejercicio anterior para que todas las letras se escriban en la misma línea, pero con la letra 'e' en un color diferente.
3. Escriba un programa que acepte como entrada su nombre y apellidos y escriba sus iniciales.
4. Escriba un programa que lea en líneas de DATA una serie de palabras y las reparta en dos columnas: en una las que empiezan con mayúscula y en otra las que empiezan con minúscula. (Recuerde que el ordenador considera que todas las letras que empiezan con mayúscula son anteriores en el orden alfabético a las que lo hacen con minúscula.)

La función INSTR

El BASIC del Amstrad dispone de una función que permite buscar rápidamente una subcadena dentro de una cadena. Se trata de la función INSTR. Por ejemplo, si escribimos

```
?instr("Rescate","es")
```

el ordenador examina la cadena 'Rescate' para averiguar si dentro de ella se encuentra la cadena 'es'. En caso afirmativo, INSTR da el número de la posición, dentro de la cadena mayor, donde empieza la subcadena (en este caso, el número 2). Escriba

```
?instr("Rescate","cate")
```

y obtendrá el número 4. ¿Pero, qué ocurre si la subcadena buscada no está en la cadena en la que se busca? La función INSTR genera entonces el número Ø:

```
?instr("Rescate","Salvamento")
```

Nada impide buscar con INSTR en cadenas mucho más largas:

```
?instr("este es el testamento","te")
```

(El único límite es el impuesto por la máxima longitud de las cadenas que admite el Amstrad: 255 caracteres.) En este último ejemplo, el ordenador ha respondido con el número 1, que es la primera posición en que aparece 'es'. Una vez detectada la primera aparición de la subcadena, podemos seguir buscándola en el resto de la cadena original. Para ello debemos especificar en qué posición debe comenzar la búsqueda. En este caso, como sabemos que 'es' aparece por primera vez en la posición 1, debemos seguir buscando a partir del segundo carácter:

```
?instr(2,"este es el testamento","es")
```

Otra vez encontramos 'es', ahora en la posición 6, luego podemos seguir buscando a partir de la 7:

```
?instr(7,"este es el testamento","es")
```

El ordenador escribe el número 13, y nosotros reanudamos la búsqueda a partir de la posición 14, y así hasta que INSTR dé el número Ø. (¿Qué podemos hacer para buscar solamente la palabra 'es', o sea, para que el ordenador no detecte estas dos letras dentro de una palabra más larga, tal como 'este' o 'testamento'?)

El siguiente programa utiliza INSTR para contar el número de letras 'e' que hay en una palabra:

```
10 MODE 1
20 INPUT "Escriba una palabra: ",palabra$
30 arranque=1
40 numerodees=0
50 continuar=1
60 WHILE continuar=1
70 posicion=INSTR(arranque,palabra$,"e")
80 IF posicion>0 THEN numerodees=numerodees+1: P
   RINT"Hay una 'e' en la posicion ";posicion:ar
   ranque=posicion+1 ELSE continuar=0
90 WEND
100 PRINT "En la palabra hay";numerodees;"letras
     'e' en total"
```

La línea 8Ø actualiza el valor de 'arranque' para que la siguiente búsqueda se realice a partir de la posición siguiente a la actual. Cuando se acaban las letras 'e', la cláusula ELSE hace 'continuar' igual a Ø y el bucle termina.

Un método análogo se puede utilizar en el juego Hangman:

```
10 MODE 1
11 REM escoger una palabra aleatoriamente
20 numerodepalabras=10
```

```
30 palabraelegida=INT(RND*numerodepalabras+1)
40 FOR palabras=1 TO palabraelegida
50 READ palabra$
60 NEXT
61 REM preparar las variables
70 longitud=LEN(palabra$)
80 respuesta$=STRING$(longitud,"-")
90 intentos=1
100 adiv$=""
101 REM dar 10 oportunidades
110 WHILE intentos<11 AND respuesta$<>palabra$
120 PEN 3
130 PRINT
140 PRINT"La palabra es ";respuesta$
150 PRINT
160 PEN 1
170 PRINT"Este es el intento numero";intentos
180 PRINT
190 INPUT"Adivina una letra: ",letra$
191 REM comprobar si esa letra aparece en la pal
    abra
200 arranque=1
210 continuar=1
220 WHILE continuar=1
230 posicion=INSTR(arranque,palabra$,letra$)
231 REM si la letra aparece, colocarla en la res
    puesta en la posicion correcta
240 IF posicion>0 THEN respuesta$=LEFT$(respuest
    a$,posicion-1)+letra$+MID$(respuesta$,posici
    on+1):arranque=posicion+1 ELSE continuar=0
241 REM continuar comprobando letras hasta que n
    o haya mas coincidencias
250 WEND
251 REM otro intento
260 intentos=intentos+1
270 WEND
280 PEN 2
290 PRINT
300 IF respuesta$=palabra$ THEN PRINT"Lo consegu
    iste!" ELSE PRINT"Era ";palabra$
310 PEN 1
320 END
330 DATA chacal,lince,antilope,elefante,tigre
340 DATA rinoceronte,iguana,avestruz,panda,balle
    na
```

Este programa contiene dos bucles WHILE, uno dentro del otro. En el capítulo siguiente estudiaremos más detenidamente esta cuestión de los "bucles anidados". Las palabras que vamos a utilizar en el juego están en las líneas de DATA, 330 y 340. Si quiere añadir otras palabras, ponga más sentencias DATA y modifique la línea 20 para que el ordenador sepa cuántas palabras hay.

La primera parte del programa selecciona una palabra al azar, para lo cual lee un número de palabras determinado aleatoriamente. Una vez elegida la palabra, la línea 80 forma una cadena de la misma longitud y compuesta por guiones. El bucle WHILE externo que empieza en la línea 110 da 10 oportunidades para adivinar la palabra. El bucle interno, líneas 220 a 250, examina la palabra para determinar en qué posición se encuentra la letra tecleada, si es que está en ella. Si la letra está en la palabra, la línea 240 coloca la letra en la cadena 'respuesta$' en la posición correcta. Es decir, cuando se acierta una letra, ésta sustituye un guión (o varios) en 'respuesta$'. El programa termina cuando las cadenas 'palabra$' y 'respuesta$' son idénticas o cuando se han consumido las 10 oportunidades.

Quizá le resulte difícil de seguir el programa. En cualquier caso, sirva como demostración de que, con las instrucciones de BASIC que hemos introducido hasta ahora, es posible escribir programas relativamente sofisticados. Esta versión de Hangman funciona igual que la incluída en la cinta de demostración, aunque sin el atractivo de los gráficos.

Ejercicios

1. El programa de Hangman tiene dos pequeños defectos. El primero es que permite que el usuario teclee más de una letra en cada intento. El segundo es que no avisa al jugador cuando teclea una letra que ya ha sido introducida antes. Corrija usted estos dos defectos. (*Sugerencia* para el segundo: ¿se podría utilizar la concatenación para mantener una lista de las letras introducidas?)

EL JUEGO DE CARACTERES DEL AMSTRAD

En un programa anterior vimos cómo podíamos eliminar de una cadena caracteres que no fueran letras, tales como el punto, la coma y los dos puntos. El método no tiene interés general, ya que, además de estos tres, hay muchos otros caracteres que pueden estar al final de una cadena y no ser letras; por ejemplo, las comillas, el punto y coma, los signos de admiración y de interrogación y el de cerrar paréntesis. Sería muy pesado tener que comprobar todos estos caracteres uno a uno. ¿No habrá una forma más fácil de averiguar inmediatamente si un carácter es una letra o no lo es?

Pues sí, la hay. Todos los caracteres que se pueden generar mediante el teclado tienen asociado un código numérico, denominado código ASCII. El margen de estos códigos va del 0 a 255. Los 32 primeros, del 0 al 31, tienen para el ordenador significados especiales, tales como "borrar la pantalla" o "hacer avanzar el cursor una posición". Los códigos del 32 al 255 representan caracteres visualizables en la pan-

talla. Una de las ventajas de este sistema de codificación reside en el hecho de que las letras están agrupadas en dos series de códigos; por eso, observando su código, es muy fácil determinar si un carácter es una letra. Por ejemplo, los códigos del 65 al 90 representan las letras mayúsculas:

```
10 MODE 1
20 FOR codigo=65 TO 90
30 PRINT CHR$(codigo);
40 NEXT
```

La línea 30 nos presenta una nueva función, CHR$, que convierte los códigos ASCII en caracteres. El código 65 representa la letra 'A', el 66 la 'B', etc. Los códigos del 97 al 122 representan las letras minúsculas. Modifique la línea 20 para comprobarlo.

Los restantes códigos representan los signos de puntuación, las cifras y muchos otros caracteres que se utilizan principalmente en juegos. Modifique la línea 20 del programa anterior para que el bucle se extienda del 32 al 255 y podrá ver la mayor parte del juego de caracteres del Amstrad.

Una advertencia: los códigos inferiores al 32 representan para el ordenador diversas órdenes (son códigos de control). Si los "imprimimos" por error, obtendremos efectos inesperados. Pruebe el programa haciendo que el bucle vaya de 0 a 255. Para volver a la normalidad, ejecute una orden de modo de pantalla, o bien reinicialice la máquina con ⟨CTRL⟩ ⟨SHIFT⟩ ⟨ESC⟩.

En realidad, los códigos del 0 al 31 también representan caracteres visibles, pero para que puedan aparecer en la pantalla tienen que ir precedidos del código ASCII número 1. El código 1 es un código de control cuya función es decirle a la máquina: "visualiza en la pantalla el código que viene a continuación, aunque sea menor que 32, y déjate de tonterias":

```
10 MODE 1
20 FOR codigo=0 TO 31
30 PRINT CHR$(1);CHR$(codigo);
40 NEXT
```

Caracteres	Códigos ASCII
Códigos especiales y de control	0-31
Espacio	32
0-9	48-57
A-Z	65-90
a-z	97-122

Figura 24. Algunos de los códigos ASCII más útiles.

Muchos de estos caracteres se pueden obtener pulsando ⟨CTRL⟩ en combinación con las teclas de letras. Compruébelo recorriendo el alfabeto.

El juego de caracteres está descrito con detalle en el Manual del Usuario.

Otra ventaja de los códigos ASCII es la facilidad con la que se restringe la entrada por el teclado a un margen predeterminado de caracteres. El siguiente programa escribe las letras mayúsculas que el usuario introduzca e ignora todos los demás caracteres. El programa consiste en un bucle infinito; para interrumpirlo hay que pulsar dos veces la tecla ⟨ESC⟩:

```
10 MODE 1
20 continuar=1
30 WHILE continuar=1
40 codigo$=INKEY$
50 IF codigo$<>"" THEN codigo=ASC(codigo$)
60 IF codigo>64 AND codigo<91 THEN PRINT codigo$
   ;
70 WEND
```

En la línea 40 la función INKEY$ examina el teclado. Si detecta una pulsación, guarda el carácter introducido en la variable 'codigo$'. En la línea 50 encontramos otra función de BASIC: ASC convierte un carácter en su correspondiente código ASCII. Necesitamos la comprobación IF ... THEN porque, si no se ha pulsado ninguna tecla, codigo$ es la *cadena vacía* (ningún carácter), y el ordenador emite un mensaje de error si le pedimos que halle el código ASCII de algo que no existe. La línea 60 escribe codigo$ si el código del carácter es mayor que 64 y menor que 91, o sea, si representa una letra mayúscula.

Podemos utilizar los códigos ASCII para escribir un sencillo programa codificador/decodificador:

```
10 PRINT
20 INPUT "Cual es el codigo";desplazamiento
30 PRINT
40 LINE INPUT "Cual es el mensaje? ",mensaje$
50 longitud=LEN(mensaje$)
60 mensajecodificado$=""
70 FOR cuenta=1 TO longitud
80 letra$=MID$(mensaje$,cuenta,1)
90 letraascii=ASC(letra$)
100 letracodificada$=CHR$(letraascii+desplazamie
    nto)
110 mensajecodificado$=mensajecodificado$+letrac
    odificada$
120 NEXT
130 PRINT
140 PRINT"El mensaje codificado es: "
150 PEN 3
```

```
160 PRINT
170 PRINT mensajecodificado$
180 PEN 1
```

Responda a la primera pregunta con un número igual o menor que 120. El ordenador capta el mensaje en la línea 40. El bucle de las líneas 70 a 120 toma una a una las letras del mensaje y suma a su código ASCII el valor de 'desplazamiento' para obtener otro código ASCII. La línea 100 convierte este código en un carácter, y la línea 110 coloca el carácter al final de la cadena 'mensajecodificado$'.

Al principio del programa no hemos puesto la habitual instrucción de modo de pantalla. De esta manera, al no borrarse la pantalla, usted puede decodificar un mensaje sin tener que recordar la versión codificada. Copie el mensaje codificado con la tecla ⟨COPY⟩ al responder a línea 40. Para decodificar correctamente un mensaje, hay que utilizar el mismo valor de 'desplazamiento' con que fue codificado, pero cambiado de signo. Por ejemplo, si codificó su mensaje con 23, decodifíquelo con −23.

Las funciones VAL y STR$

En ocasiones interesa tratar los números como si fuesen cadenas literales. Un caso típico es el de las fechas. Sin embargo, en algún lugar del programa podremos necesitar esos números en algún cálculo, y ya sabemos que el Amstrad se niega rotundamente a realizar operaciones aritméticas con cadenas literales. Así pues, tenemos que aprender a convertir números de su forma normal a forma literal, y viceversa. La función VAL convierte una cadena en un número:

```
10 MODE 1
20 INPUT "Escriba lo que quiera, pero empezando
    con un numero: ",caracter$
30 numero=VAL(caracter$)
40 PRINT "El numero era";numero
```

VAL empieza con examinar el primer carácter de la cadena. Si no es una cifra, ya no comprueba más caracteres y genera el número 0. Esto es lo que ocurre si se escribe 'Junio 6' en respuesta a la línea 20 del programa anterior. No obstante, podríamos extraer el número '6' con el siguiente programa:

```
10 MODE 1
20 INPUT "Introduzca una fecha (por ejemplo, 6 d
e junio): ",fecha$
30 continuar=1
40 posicion=1
50 WHILE continuar=1
60 letra$=MID$(fecha$,posicion,1)
70 codigo=ASC(letra$)
```

```
80 IF codigo>47 AND codigo<58 THEN continuar=0 E
   LSE posicion=posicion+1
90 WEND
100 numero$=MID$(fecha$,posicion)
110 numero=VAL(numero$)
120 PEN 3
130  PRINT "El dia era el";numero
```

El programa examina una a una las letras de la cadena hasta que encuentra un código que representa una cifra (línea 80). La línea 100 extrae entonces la subcadena en la que está el número; la línea 110 la convierte con VAL.

STR$ realiza la función inversa: convierte un número en su forma literal:

```
10 MODE 1
20 INPUT "Escriba la fecha actual (en la forma
    31/6/83): ",fecha$
21 REM convertir dia y mes a numeros
30 dia=VAL(fecha$)
40 posicionmes=INSTR(fecha$,"/")
50 mes$=MID$(fecha$,posicionmes+1)
60 mes=VAL(mes$)
70 posicionano=INSTR(posicionmes+1,fecha$,"/")
80 ano$=MID$(fecha$,posicionano+1)
81 REM averiguar el nombre del mes
90 FOR cuenta=1 TO mes
100 READ nombremes$,numerodedias
110 NEXT
111 REM averiguar la fecha de la semana siguient
    e
120 dia=dia+7
121 REM si este numero es mayor que el numero de
     dias del mes, pasar al mes siguiente
130 IF dia>numerodedias THEN dia=dia-numerodedia
    s:READ nombremes$,numerodedias
135 READ ind$:IF ind$="XXX" THEN ano$=MID$(STR$(
    VAL(ano$)+1),2)
140 dentrodeunasemana$=STR$(dia)+" de "+nombreme
    s$+" de 19"+ano$
150 PRINT
160 PRINT"Dentro de una semana la fecha sera: "
170 PEN 3
180 PRINT dentrodeunasemana$
190 PEN 1
200 DATA Enero,31,Febrero,28,Marzo,31,Abril,30,M
    ayo,31,Junio,30
210 DATA Julio,31,Agosto,31,Setiembre,30,Octubre
    ,31,Noviembre,30,Diciembre,31,Enero,31,XXX
```

Este programa toma la fecha actual y calcula la de una semana más tarde. Tanto el día como el mes tienen que estar en forma numérica: el día, para poder hacer cálculos con él; el mes, para saber cuántos datos hay que leer en el bucle de las líneas 9Ø a 12Ø. La línea 14Ø convierte la variable numérica 'dia' en variable literal.

Aunque en este momento no le parezca demasiado interesante esto de convertir variables numéricas en variables literales, y viceversa, vale la pena que estudie la técnica de extracción de valores numéricos contenidos en una cadena. La conversión de números en cadenas literales para luego concatenar las cadenas es una forma sencilla de agrupar datos interrelacionados sin tener que describirlos individualmente. Los números originales se pueden recuperar fácilmente utilizando MID$ para extraer la parte de la cadena que se desee.

Ejercicios

1. Escriba un programa que capte números por el teclado y rechace todo lo demás.
2. Un defecto del programa que calcula la fecha de dentro de una semana es que no tiene en cuenta los años bisiestos. Los años son bisiestos si son divisibles por 4, con la excepción de los años en que cambia el siglo (17ØØ, 18ØØ, etc.), que sólo son bisiestos si son divisibles por 4ØØ. Modifique el programa para que asigne a febrero el número de días correcto.
3. Escriba un programa que calcule cuántos días laborables quedan desde hoy hasta el día de Navidad.
4. Otro del mismo estilo. Escriba un programa que calcule cuántos días ha vivido el usuario, después de preguntarle la fecha de nacimiento y la fecha actual.

7

Bucles y listas

BUCLES ANIDADOS

En un ejemplo del capítulo anterior utilizábamos dos bucles, un bucle dentro del otro. Los *bucles anidados* son interesantes porque permiten simplificar los programas. Veamos un sencillo ejemplo que ilustra la naturaleza de los bucles anidados:

```
10 MODE 1
20 FOR bucleexterior=1 TO 3
30 PEN 3
40 PRINT "El bucle exterior es";bucleexterior
50 FOR bucleinterior=1 TO 4
60 PEN 2
70 PRINT "El bucle interior es";bucleinterior
80 NEXT
90 NEXT
100 PEN 1
```

La salida del programa es la que se muestra en la figura 25.

```
El bucle exterior es 1
El bucle interior es 1
El bucle interior es 2
El bucle interior es 3
El bucle interior es 4
El bucle exterior es 2
El bucle interior es 1
El bucle interior es 2
El bucle interior es 3
El bucle interior es 4
El bucle exterior es 3
El bucle interior es 1
El bucle interior es 2
El bucle interior es 3
El bucle interior es 4
```

Figura 25. Salida del programa de bucles anidados.

Cuando se ejecuta el programa, el ordenador abre el bucle exterior (línea 2Ø) y describe el valor de su variable de control, que inicialmente es 1 (línea 4Ø). El bucle interior comienza en la línea 5Ø, antes de que el programa haya salido del exterior, pues no ha llegado al correspondiente NEXT. La línea 7Ø escribe el valor de la variable de control del bucle interior, que es 1 en la primera pasada.

En la línea 8Ø aparece la primera sentencia NEXT. Tal como está organizado el lenguaje BASIC, este NEXT no corresponde al primer FOR, sino al segundo, o sea, al bucle interior. Así pues, el ordenador continúa recorriendo el bucle interior hasta que su variable de control alcanza el límite superior especificado en la línea 5Ø. Cuando completa el bucle interior, pasa a la siguiente instrucción, la de la línea 9Ø, que es otro NEXT. Esta sentencia NEXT tiene que corresponder al último FOR que no haya sido todavía emparejado con un NEXT; en otras palabras, el NEXT de la línea 9Ø señala el final del bucle abierto en la línea 2Ø.

Al llegar por primera vez a la línea 9Ø, el ordenador vuelve a la línea 2Ø para realizar la segunda pasada por el bucle exterior, escribe el número 2 (línea 4Ø) y entra otra vez en el bucle interior, escribiendo por segunda vez los números 1, 2, 3 y 4. Y así sucesivamente.

La razón por la que estos bucles se denominan "bucles anidados" se evidencia en la figura 26.

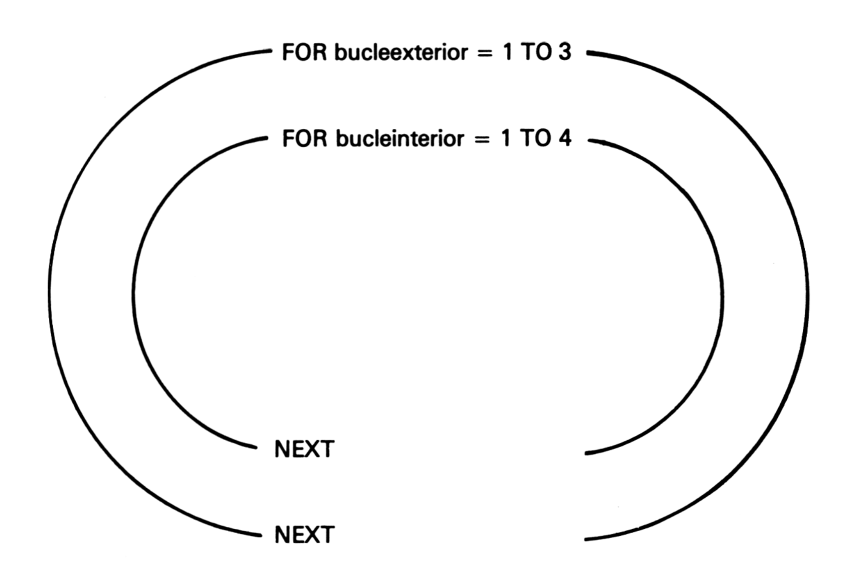


Figura 26. Bucles anidados.

Pasemos a un ejemplo más práctico, extraordinariamente corto, que no obstante escribe las tablas de multiplicar completas:

```
10 MODE 1
20 FOR tabla=1 TO 10
30 PEN 3
40 PRINT "tabla de multiplicar del";tabla
50 PRINT
60 PEN 2
70 FOR numero=1 TO 10
80 PRINT numero;"por";tabla;"es";numero*tabla
90 NEXT
100 PRINT
110 NEXT
120 PEN 1
```

Si no fuera por los bucles anidados, este programa habría sido mucho más largo, pues habríamos necesitado 10 bucles independientes para conseguir el mismo efecto. Las tablas pasan por la pantalla a velocidad excesiva; introduzca una línea con una instrucción INPUT que pregunte al usuario si ha terminado de observar una tabla y quiere ver la siguiente.

El siguiente programa dibuja una serie de rectángulos que llena la pantalla:

```
10 MODE 1
20 FOR coordenadax=0 TO 500 STEP 100
30 FOR coordenaday=0 TO 350 STEP 50
40 MOVE coordenadax,coordenaday
50 DRAW coordenadax+90,coordenaday+10
60 DRAW coordenadax+40,coordenaday+40
70 DRAW coordenadax,coordenaday
80 NEXT
90 NEXT
```

El valor del paso (STEP) y las coordenadas son arbitrarios. Experimente con otros valores.

En el capítulo 3 desarrollamos un programa que dibuja una casa. Reduciendo el tamaño de la casa y utilizando un bucle podemos dibujar una calle entera:

```
10 MODE 1
20 REM coordenadas de la fachada
30 FOR casaizquierda=0 TO 500 STEP 100
40 casaabajo=200
50 casaderecha=casaizquierda+90
60 casaarriba=250
70 REM dibujar la fachada
80 MOVE casaizquierda,casaabajo
90 DRAW casaderecha,casaabajo
100 DRAW casaderecha,casaarriba
110 DRAW casaizquierda,casaarriba
```

```
120 DRAW casaizquierda,casaabajo
130 REM coordenadas del tejado
140 tejadoizquierda=casaizquierda+15
150 tejadoarriba=270
160 tejadoderecha=casaizquierda+75
170 REM dibujar el tejado
180 MOVE casaizquierda,casaarriba
190 DRAW tejadoizquierda,tejadoarriba
200 DRAW tejadoderecha,tejadoarriba
210 DRAW casaderecha,casaarriba
220 NEXT
```

Un programa del capítulo anterior codificaba los mensajes que le entregábamos. Basándonos en esa idea hemos escrito un programa que propone una serie de 10 mensajes codificados y pide al usuario que los descifre lo más deprisa posible:

```
10 MODE 1
20 arranque=TIME
30 FOR clave=1 TO 10
40 READ palabra$
50 clave$=palabra$
60 longitud=LEN(palabra$)
70 FOR lio=1 TO longitud
80 aleat=INT(RND*longitud+1)
90 der=longitud-aleat
100 clave$=LEFT$(clave$,aleat-1)+RIGHT$(clave$,d
    er)+MID$(clave$,aleat,1)
110 NEXT
120 PEN 1
130 PRINT:PRINT "La clave es ";
140 PEN 3
150 PRINT clave$
160 PEN 2
170 PRINT:INPUT "Cual es la palabra";adiv$
180 PRINT
190 IF adiv$=palabra$ THEN PEN 3:PRINT"Bien!" EL
    SE PEN 1:PRINT"Falso, es ";palabra$
200 NEXT
210 total=TIME-arranque
220 PRINT"Ha tardado";total/300;"segundos."
230 DATA listo,enorme,canguro,medusa,mobiliario
240 DATA electricidad,ventana,poliestireno,embud
    o,macarrones
```

De los dos bucles de este programa, el exterior hace las preguntas y el interior revuelve las letras para generar los mensajes codificados. Podemos cambiar el tipo de bu-

cle; el exterior lo haremos del tipo WHILE para facilitar el control del tiempo. Añada las siguientes líneas:

```
11 limite=100
12 PRINT "Dispones de ";limite;"segundos para de
   scifrar 10 claves."
13 segundos=0
14 correcto=1
15 cuentapalabras=0
31 cuentapalabras=cuentapalabras+1
190 IF adiv$=palabra$ THEN correcto=correcto+1:P
    EN 3:PRINT"Bien!" ELSE PEN 1:PRINT"Falso, es
     ";palabra$
191 tiempoahora=TIME-arranque
192 segundos=tiempoahora/300
193 PRINT:PRINT"Llevas consumidos";segundos;"seg
    undos."
225 IF correcto>1 THEN PRINT"Tienes";correcto;"b
    ien." ELSE PRINT"No acertaste ni una!"
```

También podemos hacer que los dos bucles anidados sean del tipo WHILE:

```
10 MODE 1
20 READ pregunta$,respuestacorrecta$
30 WHILE pregunta$<>"XXX"
50 PEN 3
60 PRINT:PRINT pregunta$
70 PEN 2
80 respuesta$=""
90 intentos=1
100 WHILE respuesta$<>respuestacorrecta$ AND int
    entos<4
110 LINE INPUT respuesta$
120 intentos=intentos+1
130 IF respuesta$<>respuestacorrecta$ AND intent
    os<4 THEN PRINT"No. Intentalo otra vez."
140 WEND
150 IF respuesta$<>respuestacorrecta$ THEN PRINT
    "La respuesta era: ";respuestacorrecta$
160 READ pregunta$,respuestacorrecta$
170 WEND
180 PEN 1
190 DATA Cuantos dias tiene mayo?,31
200 DATA Cual es la capital del Reino Unido?,Lon
    dres
210 DATA Quien gano el ultimo premio individual
```

```
      masculino de Wimbledon?,John McEnroe
220 DATA Que pais gano la ultima copa del mundo
      de futbol?,Italia
500 DATA XXX,YYY
```

En este caso, el primer bucle WHILE, que va de la línea 30 a la 170, formula preguntas hasta que el dato leído es el terminador 'XXX'. El bucle interno da tres oportunidades para cada pregunta. La principal ventaja de utilizar un WHILE en lugar de un FOR ... NEXT en el bucle externo es que de esta manera podemos aumentar el número de preguntas sin más que modificar los datos y sin tener que cambiar ninguna otra línea del programa (y cuidando que los nuevos datos estén en líneas de número inferior a 500). Si el bucle fuera del tipo FOR ... NEXT, tendríamos que modificar el valor del estremo superior de la variable de control cada vez que añadiésemos una nueva pregunta.

Ejercicios

1. Dibuje un rectángulo formado enteramente por asteriscos utilizando un par de bucles anidados. (El mismo efecto se puede conseguir con STRING$ y un bucle sencillo; pero practique con los bucles anidados.)
2. Escriba un asterisco en todas las posiciones de texto que tengan impar al menos una de sus coordenadas: (1,1), (1,3), (1,5), ... , (3,1), (3,3), ...
3. Escriba un programa que lea los nombres de una serie de alumnos y sus notas en doce exámenes y que escriba toda esta información en la pantalla, junto con las notas medias. Utilice dos bucles anidados: el externo para recorrer todos los alumnos, y el segundo para leer las doce notas de cada alumno.
4. Amplie el programa que dibuja una fila de casas para que dibuje varias filas de colores distintos. Ponga puertas y ventanas en las casas.
5. Escriba un programa que dibuje un edificio de 3 plantas, con 5 ventanas por planta. Todas las ventanas deben ser del mismo tamaño.

LISTAS

Los dos últimos programas de la sección anterior tenían el mismo defecto: cada vez que se ejecuta el programa éste formula las mismas preguntas. Si intentamos resolver este problema leyendo cada vez un número aleatorio de preguntas, creamos otro nuevo: cómo garantizar que no repetimos ninguna pregunta.

En uno de los ejercicios del final de la sección anterior le pedíamos que escribiese un programa para leer los nombres y las notas de una serie de alumnos y escribirlos en la pantalla. En programa más cercano a la realidad, esta información tendría que ser utilizada más tarde para otros fines; por ejemplo, para elaborar una lista de los nombres ordenada de mayor a menor nota en Matemáticas. También podríamos necesitar repetir esta operación para las 11 asignaturas restantes. Ahora bien,

para ordenar las notas, el programa necesita conocer todas la de una asignatura al mismo tiempo.

Todo esto sugiere la necesidad de que el ordenador pueda manejar *listas* de datos, de forma que pueda comparar unos con otros. Con nuestros conocimientos actuales, esta tarea es enormemente tediosa, cuando no imposible. Supongamos que tenemos quince alumnos y que tenemos que ordenarlos según sus notas de Matemáticas. El ordenador necesita conocer simultáneamente las quince notas para poder compararlas y ordenarlas. La primera parte del programa tendría que leer los nombres y las notas:

```
10 MODE 1
20 READ nombre1$,nota1
30 READ nombre2$,nota2
40 READ nombre3$,nota3
50 READ nombre4$,nota4
60 READ nombre5$,nota5
70 READ nombre6$,nota6
80 READ nombre7$,nota7
90 READ nombre8$,nota8
100 READ nombre9$,nota9
110 READ nombre10$,nota10
120 READ nombre11$,nota11
130 READ nombre12$,nota12
```

y así sucesivamente, y ni siquiera hemos empezado a comparar notas. ¿Se imagina qué habría que hacer si los alumnos fueran 100? Este método es impracticable.

En lugar de guardar cada dato en una variable, lo que necesitamos es una variable de un tipo especial, una lista, que contenga todos los datos:

```
10 MODE 1
20 DIM nombre$(15),nota(15)
30 FOR cuenta=1 TO 15
40 READ nombre$(cuenta),nota(cuenta)
50 NEXT
400 DATA Alvarez,4.0,Benito,5.6,Cuesta,7.7,Diegu
    ez,2,Martin,8.4
410 DATA Fernandez,4.5,Garcia,9,Hernandez,3,Medi
    na,9.5,Nieto,9.5
420 DATA Ruiz,0,Sanchez,5.5,Tomas,6,Victor,4.5,Z
    acarias,8
```

La línea 20 informa al ordenador de cuántos elementos va a haber en cada lista: 15. La palabra clave de BASIC, DIM, *dimensiona* las listas. Observe que las listas pueden ser literales, nombre$( ), o numéricas, nota( ). El bucle de las líneas 30 a 50 lee los datos y los guarda en las dos listas. El número que figura entre paréntesis en la línea 40 es lo que se denomina *subíndice*. El subíndice varía al mismo tiempo

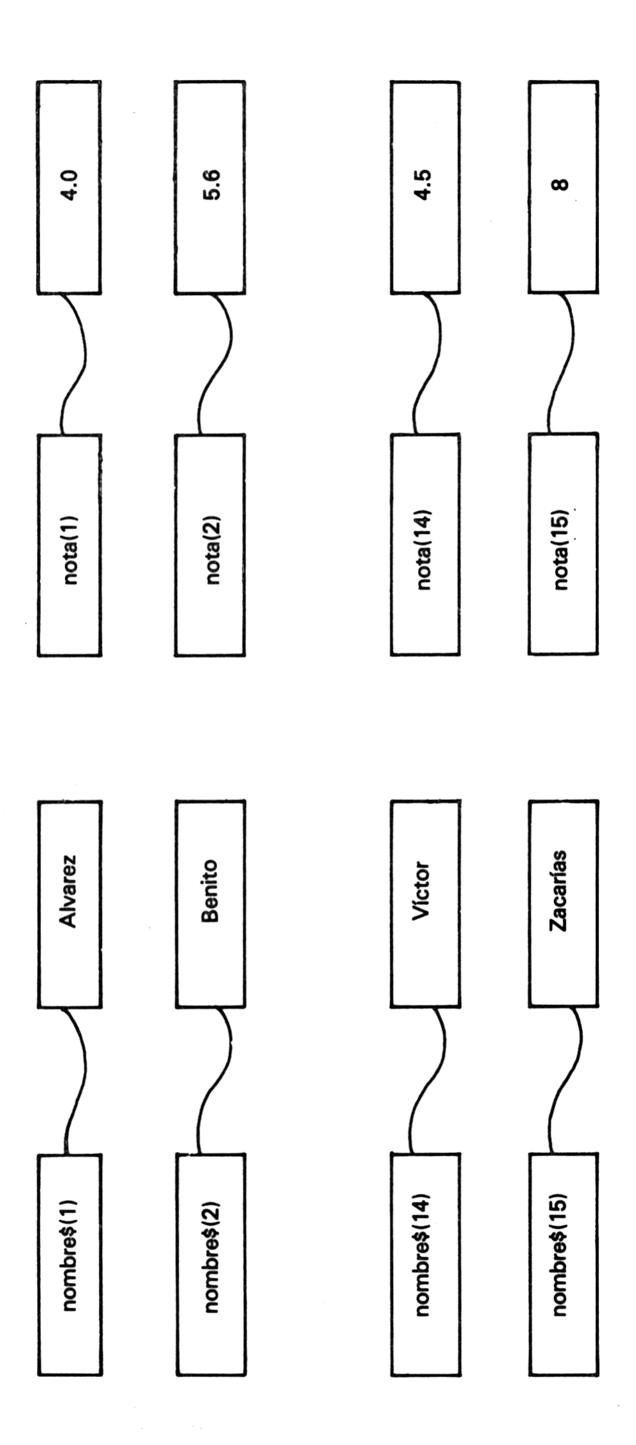


Figura 27. Almacenamiento de datos en listas.

que la variable de control del bucle. El resultado final es que, cuando el bucle termina, todos los datos han quedado guardados en las listas, según se muestra en la figura 27.

Ahora podemos acceder a los diferentes elementos de la lista mencionándolos por su subíndice. Después de ejecutar el programa, pruebe la siguiente orden en modo directo:

```
nombre$(15)
```

El ordenador escribe 'Zacarías', que es el decimoquinto elemento de la lista 'nombre$'. Escriba

```
nota(5)
```

El ordenador escribe '8.4', que es la nota del alumno número 5. Una vez guardados los datos en las listas, podemos hacer que el ordenador los manipule a nuestro antojo sin tener que volver a leerlos en las líneas de DATA. Añada las siguientes líneas al programa:

```
2 MODE 1
60 PRINT "Voy a escribir los nombres de los
     alumnos que tienen nota inferior a la que u
   sted indique."
70 PRINT
80 INPUT "Cual es la nota tope";notatope
90 PEN 3
100 PRINT:PRINT "Alumnos con nota menor que";not
    atope
110 PRINT:PRINT "Nombre","nota"
120 PEN 2
130 FOR cuenta=1 TO 15
140 IF nota(cuenta)<notatope THEN PRINT nombre$(
    cuenta),nota(cuenta)
150 NEXT
160 PEN 1
```

Al recorrer 15 veces el bucle de las líneas 130 a 150, el programa examina las notas de todos los alumnos y excribe el nombre y la nota de aquéllos cuya nota no alcanza el valor especificado. La línea 140 demuestra lo fácil que es conectar una lista con otra. Por ejemplo, nombre$(14) es el nombre del 14 alumno; para averiguar su nota no tenemos más que consultar nota(14). Los componentes individuales de la lista, tales como nombre$(14) o nota(5), son sus *elementos*.

Aparte la necesidad de mencionar un subíndice, por todo lo demás el ordenador trata los elementos de las listas como si fueran variables ordinarias (numéricas o literales, según el caso). Pruebe, por ejemplo, las siguentes órdenes directas:

```
?LEFT$(nombre$(11),2)
?nota(7)+nota(12)
?nombre$(1)+nombre$(6)
```

Los elementos de las listas no se distinguen de las variables ordinarias más que en la inclusión del subíndice.

Hemos utilizado listas para guardar datos en ellas. Pero también nos pueden servir para almacenar resultados. En un programa anterior simulábamos el lanzamiento de dos dados generando números aleatorios. Ahora ya podemos almacenar el resultado de cada lanzamiento en los elementos de una lista, para luego utilizarlos según nos interese:

```
10 MODE 1
20 WINDOW 1,40,1,1
30 WINDOW #1,1,40,2,25
40 INPUT "Cuantas tiradas";tiradas
50 DIM resultado(12)
60 FOR totaldados=2 TO 12
70 LOCATE#1,1,totaldados*2
80 PRINT#1,totaldados
90 NEXT
100 FOR cuenta=1 TO tiradas
110 dado1=INT(RND*6+1)
120 dado2=INT(RND*6+1)
130 total=dado1+dado2
140 resultado(total)=resultado(total)+1
150 LOCATE#1,4,total*2
160 PRINT#1,resultado(total)
170 NEXT
```

Siempre que se dimensiona una lista, como en la línea 5Ø de este programa, los elementos de la lista se igualan a cero (o a la cadena vacía si la lista es literal). Lo que hemos hecho en esa línea es preparar 13 cajas vacías, en las que más tarde pondremos los valores a medida que los vayamos generando. (Hay dos cajas que no vamos a utilizar, resultado(Ø) y resultado(1), porque al lanzar dos dados el resultado no puede ser menor que 2.)

Las líneas 6Ø a 9Ø escriben los números del 2 al 12, convenientemente espaciados para facilitar la visualización de los resultados.

Cada vez que lanzamos los dados, las líneas 11Ø a 13Ø generan los resultados de la tirada; la línea 14Ø actualiza el valor del elemento correspondiente de la lista 'resultado( )'. La línea 16Ø escribe el nuevo valor del elemento de la lista en la posición de la pantalla que le corresponde. Si esta forma de visualización le parece ''sosa'', modifique la línea 16Ø:

```
160 PRINT#1,STRING$(resultado(total),"*")
```

Esta línea escribe una cadena de asteriscos cuya longitud es igual al número de veces que ha aparecido el resultado. El programa es más atractivo si visualizamos la distribución de los resultados con barras rectangulares, para lo cual necesitamos instrucciones de dibujo:

```
100 FOR cuenta=1 TO tiradas
110 dado1=INT(RND*6+1)
120 dado2=INT(RND*6+1)
130 total=dado1+dado2
140 resultado(total)=resultado(total)+1
141 grafy=(24-total*2)*16
142 grafx=50
143 MOVE grafx+resultado(total)*2,grafy
144 DRAW grafx+resultado(total)*2,grafy+16
170 NEXT
```

Cada vez que se genera un resultado, las líneas 143 y 144 dibujan una nueva barra. Así, el diagrama va creciendo hacia la derecha a medida que avanza el programa.

Dado que la resolución gráfica en el eje vertical es independiente del modo de pantalla, podemos hacer el diagrama mucho más atractivo dibujando las barras en distintos colores. Cambie las siguientes líneas:

```
10 MODE 0
142 grafx=150
143 MOVE grafx+resultado(total)*2,grafy
144 DRAW grafx+resultado(total)*2,grafy+16,total
```

Cada barra tiene ahora su propio color. Hemos tenido que mover la coordenada 'grafx' hacia la derecha, porque en modo Ø los números que hemos escrito junto al margen izquierdo son más anchos.

Vamos a visitar a nuestro viejo amigo, el propietario del restaurante, para ver cómo van sus intentos de informatizar el negocio. Gracias a las listas, ahora ya le es más fácil calcular las facturas. Lo primero que debe hacer es formar listas con los nombres de los artículos y los precios:

```
10 MODE 1
20 numerogeneros=12
30 DIM nombregenero$(numerogeneros),precio(numer
   ogeneros)
40 FOR cuenta=1 TO numerogeneros
50 READ nombregenero$(cuenta),precio(cuenta)
60 NEXT
400 DATA patatas p,50,patatas m,60,patatas g,70,
    bacalao p,90,bacalao m,110,bacalao g,130
410 DATA salchicha,40,flan,70,muslo pollo,140,ha
    mburguesa,100,pastel,60,perrito cal,50
```

Observe que en la línea 30 hemos utilizado una variable para especificar el tamaño de las listas. En BASIC, todos los números se pueden sustituir por variables, a excepción de los números de línea.

Una vez leídos los datos, el propietario del restaurante puede dejar el ordenador funcionando en un bucle infinito. El programa es suficientemente complejo como para que valga la pena planificarlo. Necesitamos 3 bucles del tipo WHILE anidados:

WHILE (mientras) el restaurante esté abierto
preguntar el nombre del primer artículo de la factura
WHILE (mientras) el artículo sea distinto de 'XXX'
buscar el precio del artículo
WHILE (mientras) queden elementos por examinar en la lista
continuar examinándola
If (si) se encuentra el artículo THEN (entonces) calcular el importe
WEND
IF (si) no se encuentra el artículo THEN (entonces) informar al cajero
preguntar el nombre del siguiente artículo
WEND
escribir el importe total de la factura
WEND

Este plan no es exactamente un programa, sino un *ordinograma* que conduce al siguiente programa:

```
10 MODE 1
20 numerogeneros=12
30 DIM nombregenero$(numerogeneros),precio(numer
   ogeneros)
40 FOR cuenta=1 TO numerogeneros
50 READ nombregenero$(cuenta),precio(cuenta)
60 NEXT
61 REM preparar ventanas para entradas y factura
70 WINDOW 1,40,21,25
80 WINDOW #1,1,40,1,20
81 REM primer bucle - sin fin
90 continuar=1
100 WHILE continuar=1
110 CLS
120 PEN 1
130 PRINT "Para terminar la factura introduzca X
    XX como nombre del articulo."
140 PRINT
160 INPUT "Introduzca el nombre del articulo y
        luego el numero de raciones: ",genero$,ra
    ciones
170 CLS#1
180 PEN #1,3
```

```
190 PRINT #1,"Articulo","Raciones","Precio"
200 PRINT #1
201 REM segundo bucle - para procesar cada artic
    ulo
210 WHILE genero$<>"XXX"
220 cuenta=1
230 hallado=0
231 REM buscar precio del articulo mientras no l
    o hayamos encontrado y todavia queden articu
    los en la lista
240 WHILE cuenta<=numerogeneros AND hallado=0
250 IF genero$=nombregenero$(cuenta) THEN precio
    =raciones*precio(cuenta):PRINT #1,nombregene
    ro$(cuenta),raciones,precio:preciototal=prec
    iototal+precio:hallado=1
260 cuenta=cuenta+1
270 WEND
271 REM pedir nueva introduccion del articulo si
     no lo hemos encontrado
280 IF cuenta>numerogeneros AND hallado=0 THEN P
    EN 2:PRINT"Por favor, escriba otra vez los d
    atos.":PEN 1
290 INPUT "Introduzca el nombre del articulo y
       luego el numero de raciones: ",genero$,ra
    ciones
300 WEND
301 REM calcular importe total con impuestos
310 PEN #1,2
320 PRINT #1,"Precio total:",preciototal
330 PRINT #1,"Impuestos:",,0.12*preciototal
340 PRINT #1,"Precio con impuestos:",preciototal
    *1.12
341 REM empezar factura siguiente
350 WEND
400 DATA patatas p,50,patatas m,60,patatas g,70,
    bacalao p,90,bacalao m,110,bacalao g,130
410 DATA salchicha,40,flan,70,muslo pollo,140,ha
    mburguesa,100,pastel,60,perrito cal,50
```

Este programa constituye un buen ejemplo de utilización de bucles anidados y listas. Hay un detalle que dificulta su utilización y lo hace más complicado: el usuario debe escribir el nombre de los artículos tal como éstos están en la lista de nombres, ya que el ordenador compara el nombre introducido con los elementos de la lista para localizar los artículos. El programa es mucho más sencillo si el cajero introduce el número del artículo en lugar del nombre.

Por ejemplo, 'patatas p' sería el número 1, y así salchicha' el número 7. Borre las líneas 160 y 210-350 y sustitúyalas por las siguientes:

```
160 INPUT "Introduzca el numero del articulo y
       luego el numero de raciones: ",genero,rac
    iones
210 WHILE genero<>-99
220 IF genero<>-99 AND genero<=numerogeneros THE
    N precio=raciones*precio(genero):PRINT #1,no
    mbregenero$(genero),raciones,precio:precioto
    tal=preciototal+precio
230 IF genero>numerogeneros THEN PEN 2:PRINT"Dem
    asiado grande para ser el numero     de un a
    rticulo.":PRINT"Escribalo otra vez." :PEN 1
240 INPUT "Introduzca el numero del articulo y
       luego el numero de raciones: ",genero,rac
    iones
250 WEND
260 PEN #1,2
270 PRINT #1,"Precio total:",preciototal
280 PRINT #1,"Impuestos:",0.12*preciototal
290 PRINT #1,"Precio con impuestos:",preciototal
    *1.12
300 WEND
```

Ahora se utiliza el número del artículo para buscar directamente su nombre y su precio unitario, con lo que se evita el engorro de tener que recorrer la lista en busca del nombre. Aun más sencillo sería prescindir de los nombres e ir directamente a los precios, pero entonces la factura final no es tan clara para el cliente y es más difícil detectar errores en ella.

Recomendamos al lector que estudie atentamente este programa, aunque no tenga interés práctico para él. La técnica de búsqueda de un elemento en una lista es de interés general, como tendremos ocasión de comprobar más adelante, en el capítulo que trata de ficheros.

Ejercicios

1. Modifique el programa codificador de mensajes para que haga lo siguiente: debe leer las palabras que van a ser codificadas y asignarlas a los elementos de una lista; la palabra que va a ser propuesta al jugador debe ser elegida aleatoriamente de entre los elementos de la lista. Si se siente capaz, trate de mejorar el programa para que no repita ninguna pregunta.
2. Escriba un programa que empiece por leer los nombres y números de teléfono de un grupo de amigos y los almacene en dos listas. El programa debe solicitar por el teclado el nombre de un amigo, buscar el número de teléfono y escribirlo en la pantalla, o bien emitir un mensaje adecuado si no encuentra el nombre en la lista.

3. Escriba un programa que simule el lanzamiento de una moneda generando números aleatorios (que sólo podrán tener dos valores; por ejemplo, 1 y 2). Si el número es 1, considere que el resultado es "cara"; si es 2, "cruz". Visualice los resultados en la pantalla escribiendo 'C' o 'X' para representar "cara" y "cruz", o bien dibujando un diagrama de barras.
4. Forme dos listas, una con nombres y otra con verbos, y genere frases aleatorias con la estructura "NOMBRE VERBO NOMBRE"; por ejemplo, "Los leones comen cebras". El programa se puede complicar considerablemente si se pretende manejar correctamente el género y número en los nombres y la persona en los verbos.
5. Si es realmente ambicioso, trate de diseñar algún generador aleatorio de rimas.

8

Juegos y gráficos

CARACTERES A LA CARTA

En el capítulo 6 hablamos del juego de caracteres del Amstrad. Aunque son muchos y están bien pensados, obviamente no pueden satisfacer todas las necesidades. Afortunadamente, el Basic del Amstrad ofrece una instrucción con la que se puede diseñar caracteres nuevos; la palabra clave es SYMBOL.

Todos los caracteres están basados en una retícula de 8 × 8 puntos. Para diseñar uno nuevo, lo primero que tenemos que hacer es sombrear en una hoja de papel cuadriculado los cuadraditos necesarios para lograr el efecto deseado. En la figura 28 se muestra el diseño de un extraterrestre con tentáculos.

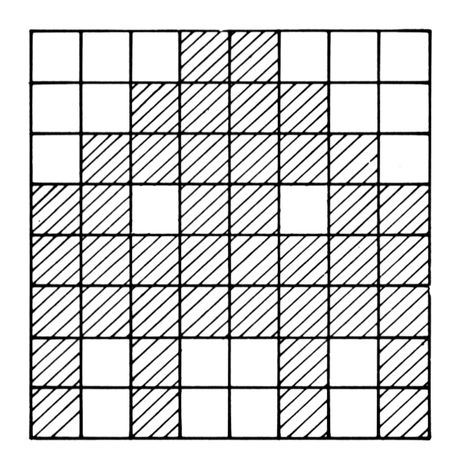


Figura 28. Diseño de un carácter para juegos.

Cada fila del carácter se describe mediante un número. Es costumbre utilizar en estos casos los números hexadecimales o los binarios; estos sistemas de numeración están descritos en el apéndice II del Manual del Usuario. Por si el lector no estuviera familiarizado con ellos, vamos a arreglárnoslas con los conocidos números decimales.

Los números que describen las filas del carácter se calculan de la siguiente forma: el número es inicialmente cero; se recorren los puntos de la fila; por cada uno que

esté sombreado, se suma el número que está en la cabecera de la columna correspondiente; si un punto no está sombreado, no se suma nada. Por este método se han calculado los números de la figura 29. Por ejemplo, en la primera fila, están sombreados los puntos de las columnas "16" y "8"; por consiguiente, el número para esta fila es 16+8=24. Y análogamente para las restantes filas.

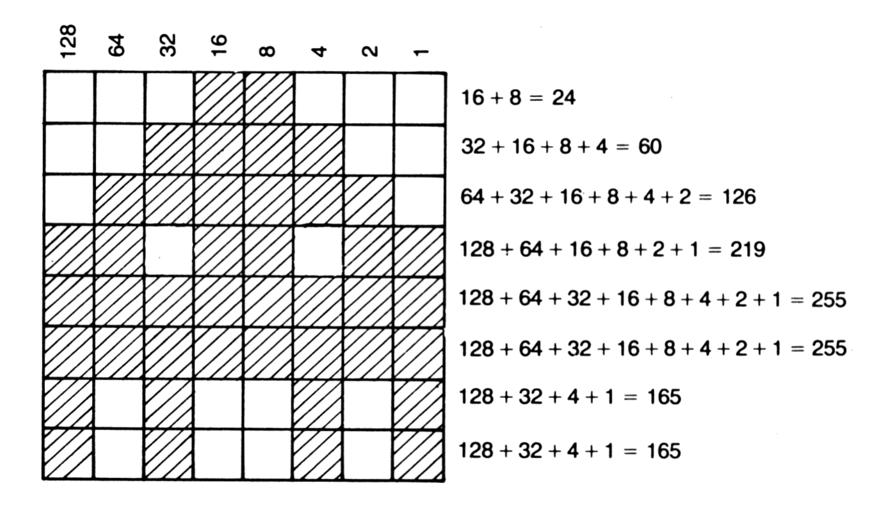


Figura 29. El extraterrestre con tentáculos, codificado.

Una vez calculados los ocho números, podemos definir el carácter y utilizarlo en los programas:

```
10 MODE 1
20 SYMBOL 240,24,60,126,219,255,255,165,165
```

La línea 20 realiza la definición del carácter. El primer número que se pone después de la palabra clave SYMBOL es el código ASCII del carácter que vamos a definir; en este caso, 240. Escriba:

```
?CHR$(240)
```

y compruebe que la definición sigue en vigor incluso después de terminar el programa. Según el apéndice III del Manual del Usuario, el carácter 240 es inicialmente una flecha que apunta hacia arriba, pero nosotros lo hemos redefinido. El Amstrad permite diseñar por este método, sin más preparativos, los caracteres del 240 al 255.

Si queremos definir otros caracteres de fuera de ese margen, necesitamos otra instrucción. Por ejemplo,

```
SYMBOL AFTER 60
```

permite definir cualquier carácter cuyo código ASCII sea igual o mayor que 6Ø. (*Symbol after* significa "símbolo después de"; pero observe que no sólo podemos definir los caracteres posteriores al 6Ø, sino también el 6Ø.) Después de ejecutar esta orden SYMBOL AFTER, podemos rediseñar casi 2ØØ caracteres a nuestro gusto.

SYMBOL AFTER anula las anteriores definiciones de caracteres. Escriba ?CHR$(24Ø) y verá que el carácter 24Ø es otra vez la flecha original.

Con SYMBOL AFTER 32 podemos redefinir cualquier carácter (incluso el espacio, que es el número 32). La siguiente instrucción cambia el diseño de la letra 'A' y le da la forma de nuestro extraterrestre:

```
SYMBOL 65,24,6Ø,126,219,255,255,165,165
```

Si usted hace lo mismo con unas cuantas letras, pronto llegará a no entender lo que escriba en la pantalla. Siempre puede volver a la situación normal ejecutando la orden SYMBOL AFTER 24Ø.

En la práctica, los 16 caracteres del margen 24Ø a 255 son suficientes para la mayor parte de las aplicaciones.

El siguiente programa demuestra con qué facilidad podemos controlar a través del teclado los movimientos del extraterrestre:

```
1Ø MODE 1
2Ø PRINT"El monstruo se puede guiar utilizando
    las teclas 'a' y 'z' para moverlo arribay ab
   ajo, y las teclas ',' y '.' para    moverlo a
    izquierda y derecha."
25 PRINT:PRINT"Pulse 'f' para parar."
3Ø INPUT "Pulse ENTER para empezar.",comienzo$
4Ø MODE Ø
5Ø SYMBOL 24Ø,24,6Ø,126,219,255,255,165,165
51 REM posicion inicial
6Ø coordx=1Ø
7Ø coordy=12
8Ø nuevacoordx=1Ø
9Ø nuevacoordy=12
1ØØ PEN 6
11Ø PAPER 1
12Ø CLS
13Ø respuesta$=""
14Ø continuar=1
15Ø WHILE respuesta$<>"f"
151 REM examinar el teclado
16Ø respuesta$=INKEY$
161 REM actualizar posicion del monstruo - compr
    obar que no esta fuera de la pantalla
17Ø IF respuesta$="a" AND coordy>1 THEN nuevacoo
    rdy=coordy-1
```

```
180 IF respuesta$="z" AND coordy<25 THEN nuevaco
    ordy=coordy+1
190 IF respuesta$="," AND coordx>1 THEN nuevacoo
    rdx=coordx-1
200 IF respuesta$="." AND coordx<20 THEN nuevaco
    ordx=coordx+1
201 REM si se ha pulsado una tecla de movimiento
    , borrar la posicion antigua
210 IF respuesta$<>"" AND respuesta$<>"s" THEN L
    OCATE coordx,coordy:PRINT" ":coordx=nuevacoo
    rdx:coordy=nuevacoordy
211 REM dibujar monstruo
220 LOCATE coordx,coordy
230 PRINT CHR$(240);
240 WEND
250 PEN 1
260 PAPER 0
```

Esta rutina forma la base de muchos programas de juegos, como veremos más adelante. Hay un par de detalles que vale la pena observar. Primero, que necesitamos cerciorarnos, antes de cada movimiento, de que el extraterrestre no se va a salir de la pantalla. En segundo lugar, tenemos que borrar el carácter de la posición actual antes de escribirlo en la siguiente. El signo de punto y coma que hemos puesto después de PRINT CHR$(240) es imprescindible. Si lo omitiésemos, el contenido de la pantalla se desplazaría hacia arriba cuando el extraterrestre llegase al estremo inferior derecho.

Las teclas que hemos utilizado para controlar el movimiento son las tradicionales en los juegos de ordenador. Si utilizásemos las teclas del cursor, tendríamos que jugar con una sola mano. En cambio, de esta manera, podemos dedicar la mano izquierda al control de los movimientos horizontales con las teclas 'z' y 'x', y la derecha a los movimientos verticales, teclas ';' y '/'. En cualquier caso, el programa es muy fácil de modificar si se prefiere otra combinación de teclas.

Como mencionábamos antes, el carácter 240 es normalmente un flecha que apunta hacia arriba. De hecho, los caracteres 240 a 243 son flechas que apuntan en las cuatro direcciones. Podríamos modificar el programa para crear cuatro caracteres que mirasen en las cuatro direcciones, y escribir el apropiado en función de la dirección del movimiento. Por no complicar las cosas, tomaremos los caracteres 240 a 243 y cambiaremos el programa para que elija el carácter adecuado a la dirección del moviento. (Antes de modificar el programa, ejecute la orden SYMBOL AFTER 240 para anular la anterior definición de este caracter.)

```
10 MODE 1
50 flecha$=CHR$(240)
170 IF respuesta$="a" AND coordy>1 THEN nuevacoo
    rdy=coordy-1:flecha$=CHR$(240)
180 IF respuesta$="z" AND coordy<25 THEN nuevaco
```

```
    ordy=coordy+1:flecha$=CHR$(241)
190 IF respuesta$="," AND coordx>1 THEN nuevacoo
    rdx=coordx-1:flecha$=CHR$(242)
200 IF respuesta$="." AND coordx<20 THEN nuevaco
    ordx=coordx+1:flecha$=CHR$(243)
230 PRINT flecha$
```

Caracteres más grandes

Los caracteres aislados pueden resultar demasiado pequeños, pero es muy fácil combinarlos para formar otros mayores. Podemos hacer una versión ampliada del extraterrestre de los tentáculos definiendo cuatro caracteres y uniéndolos, como se muestra en la figura 3Ø.

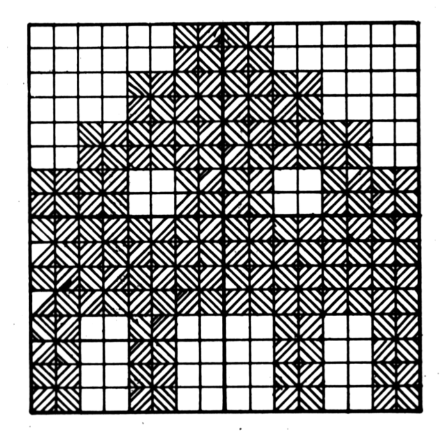


Figura 30. El extraterrestre, formado por cuatro caracteres.

Al manejar cuatro caracteres, en principio tendríamos que escribirlos en cuatro posiciones distintas. Una simplificación puede ser combinar los dos caracteres de la fila superior para formar una cadena literal, y hacer lo mismo con los dos de abajo. De este modo sólo tenemos que ocuparnos de instrucciones LOCATE:

```
10 MODE 1
20 PRINT"El monstruo se puede guiar utilizando
    las teclas 'a' y 'z' para moverlo arribay ab
   ajo, y las teclas ',' y '.' para    moverlo a
    izquierda y derecha."
25 PRINT:PRINT"Pulse 'f' para parar."
30 INPUT "Pulse ENTER para empezar.",comienzo$
40 MODE 0
41 SYMBOL 240,3,3,15,15,63,63,243,243
42 SYMBOL 241,192,192,240,240,252,252,207,207
```

```
43 SYMBOL 242,255,255,255,255,204,204,204,204
44 SYMBOL 243,255,255,255,255,51,51,51,51
45 arriba$=CHR$(240)+CHR$(241)
46 abajo$=CHR$(242)+CHR$(243)
51 REM posicion inicial
60 coordx=10
70 coordy=12
80 nuevacoordx=10
90 nuevacoordy=12
100 PEN 6
110 PAPER 1
120 CLS
130 respuesta$=""
140 continuar=1
150 WHILE respuesta$<>"f"
151 REM examinar el teclado
160 respuesta$=INKEY$
161 REM actualizar posicion del monstruo - compr
    obar que no esta fuera de la pantalla
170 IF respuesta$="a" AND coordy>1 THEN nuevacoo
    rdy=coordy-1
180 IF respuesta$="z" AND coordy<24 THEN nuevaco
    ordy=coordy+1
190 IF respuesta$="," AND coordx>1 THEN nuevacoo
    rdx=coordx-1
200 IF respuesta$="." AND coordx<19 THEN nuevaco
    ordx=coordx+1
201 REM si se ha pulsado una tecla de movimiento
    , borrar la posicion antigua
210 IF respuesta$<>"" AND respuesta$<>"s" THEN L
    OCATE coordx,coordy:PRINT"  ":LOCATE coordx,
    coordy+1:PRINT"  ":coordx=nuevacoordx:coordy
    =nuevacoordy
211 REM dibujar monstruo
220 LOCATE coordx,coordy
230 PRINT arriba$;
231 LOCATE coordx,coordy+1
232 PRINT abajo$;
240 WEND
250 PEN 1
260 PAPER 0
```

En las líneas 180 y 200 hemos simplificado los valores máximos de las coordenadas de texto. Esto se debe al tamaño del caracter.

Ejercicios

1. Diseñe un carácter y visualícelo en la pantalla con las 16 plumas disponibles en modo Ø.
2. Modifique el programa que mueve las flechas por la pantalla para que permita el movimiento en diagonal. Defina otros cuatro caracteres y haga que el programa dibuje el adecuado a cada dirección de movimiento.
3. Diseñe un carácter a su gusto (por ejemplo, un perro o un robot) y escriba un programa que permita controlar mediante el teclado sus movimientos por la pantalla.

Rebotes y disparos

Ya disponemos de casi todos los elementos necesarios para programar un juego. Sabemos cómo definir caracteres y cómo moverlos por la pantalla impidiendo que se salgan de ella. Pero aún tenemos que aprender algo más. En todos los juegos en los que hay movimientos por la pantalla, necesitamos poder detectar qué hay al lado del objeto móvil. Por ejemplo, si vamos a disparar sobre un ejército invasor, tendremos que saber si hay un enemigo en la trayectoria de nuestros proyectiles. Si estamos tratando de salir de un laberinto, tenemos que saber detectar las vías libres, pues el juego pierde todo su sentido si nos permite atravesar las paredes.

Con la función TEST(x,y) podemos detectar con qué pluma se ha dibujado una posición gráfica de la pantalla. En el programa anterior trabajábamos con coordenadas de texto, pero en el capítulo 3 hemos desarrollado la fórmula que relaciona las coordenadas de texto con las coordenadas gráficas:

$$\text{graficosx} = (\text{textox} - 1) \times 32 \qquad \text{(ponga 16 para modo 1 y 8 para modo 2)}$$
$$\text{graficosy} = (25 - \text{textoy}) \times 16$$

Estas fórmulas dan las coordenadas gráficas del punto extremo inferior izquierdo del carácter en función de las coordenadas de texto. Las coordenadas gráficas del *centro* (aproximado) del carácter serían:

$$\text{graficosx} = 16 + (\text{texto} - 1) \times 32 \qquad \text{(8 y 16 en modo 1; 4 y 8 en modo 2)}$$
$$\text{graficosy} = 8 + (25 - \text{textoy}) \times 16$$

Ahora ya podemos averiguar el color que hay en la posición de texto que necesitamos comprobar. Veamos un ejemplo que está basado en el programa de movimiento de flechas:

```
10 MODE 1
20 PRINT "Debe llevar la flecha al rincon inferi
   orderecho con las teclas 'a' 'z' ',' '.' lo m
   as deprisa posible."
30 PRINT:INPUT "Nivel de habilidad (1-10) (1 es
   facil;  10 es dificil)";habilidad
```

```
40 PRINT:INPUT "Pulse ENTER para empezar.",comie
   nzo$
50 MODE 0
60 PAPER 1
70 CLS
80 flecha$=CHR$(240)
90 REM posicion inicial
100 coordx=1
110 coordy=1
120 REM paredes rojas
130 PEN 3
140 FOR cuenta=1 TO habilidad*12
150 aleatx=INT(RND*20+1)
160 aleaty=INT(RND*25+1)
170 LOCATE aleatx,aleaty
180 PRINT CHR$(233);
190 NEXT
200 REM objetivo en color parpadeante
210 PEN 15
220 LOCATE 20,25
221 PRINT "*";
230 PEN 6
240 respuesta$=""
250 REM bucle que continua hasta que se llega a
    (20,25)
260 WHILE coordx<>20 OR coordy<>25
270 REM examinar el teclado
280 respuesta$=INKEY$
290 REM poner coordenadas en la posicion actual
300 nuevacoordx=coordx
310 nuevacoordy=coordy
320 REM actualizar posicion de la flecha - compr
    obar que no esta fuera de la pantalla
330 IF respuesta$="a" AND coordy>1 THEN nuevacoo
    rdy=coordy-1:flecha$=CHR$(240)
340 IF respuesta$="z" AND coordy<25 THEN nuevaco
    ordy=coordy+1:flecha$=CHR$(241)
350 IF respuesta$="," AND coordx>1 THEN nuevacoo
    rdx=coordx-1:flecha$=CHR$(242)
360 IF respuesta$="." AND coordx<20 THEN nuevaco
    ordx=coordx+1:flecha$=CHR$(243)
370 grafx=16+(nuevacoordx-1)*32
380 grafy=8+(25-nuevacoordy)*16
390 color=TEST(grafx,grafy)
400 REM si es una pared, no mover la flecha
```

```
410 IF respuesta$<>"" AND color<>3 THEN LOCATE c
    oordx,coordy:PRINT" ";:coordx=nuevacoordx:co
    ordy=nuevacoordy
420 REM dibujar flecha
430 LOCATE coordx,coordy
440 PRINT flecha$;
450 WEND
460 PEN 1
470 PAPER 0
```

El programa dibuja un laberinto que consiste en una serie de obstáculos rojos. El jugador debe guiar la flecha por entre los obstáculos y llevarla lo más deprisa posible al rincón inferior derecho de la pantalla. Usted puede variar la dificultad del juego; el número de obstáculos (líneas 140 a 190) depende del nivel de habilidad (línea 30).

Los obstáculos se dibujan con la pluma número 3. La línea 410 examina el color de la futura posición de la flecha; si ese color es distinto del de la pluma 3, escribe la flecha en la nueva posición.

El juego no está todavía completo, pues puede ocurrir que los obstáculos bloqueen todos los caminos. Esto tiene varias soluciones. Una es impedir que se pongan obstáculos en las inmediaciones de los rincones superior izquierdo e inferior derecho, pues de esta forma se reduce la probabilidad de que haya bloqueo. Otra posibilidad es dejar que el jugador pueda volar cierto número de obstáculos:

```
75 voladuras=0
410 IF respuesta$<>"" AND (color<>3 OR voladuras
    <2) THEN LOCATE coordx,coordy:PRINT" ";:coor
    dx=nuevacoordx:coordy=nuevacoordy
415 IF respuesta$<>"" AND color=3 THEN voladuras
    =voladuras+1
```

En esta versión el jugador puede destruir dos obstáculos atravesándolos con la flecha. La condición de la línea 410 nos ha quedado relativamente complicada: si se ha pulsado una tecla de movimiento **y además,** o bien la próxima posición está libre **o bien** el jugador todavía tiene derecho a volar un obstáculo, **entonces** mover la flecha.

Estos mismos principios se pueden aplicar en muchos otros juegos. El jugador puede guiar un coche de carreras por una pista, evitando las manchas de aceite; o bien guiar la oruga hacia las hojas verdes evitando las moras venenosas. En ambos casos, el programa debe averiguar el color de la próxima posición de texto y decidir qué hacer en función del color que haya detectado.

Veamos otro ejemplo de aplicación de TEST. Se trata de un juego para dos jugadores; cada jugador debe evitar chocar con las paredes y contra el otro jugador. Los caracteres van dejando un rastro según se mueven, de modo que el juego se va haciendo progresivamente más difícil.

En un juego de este tipo lo más sencillo es dividir el programa en varias secciones. Empezaremos por las instrucciones de juego:

```
10 MODE 1
20 PEN 1
30 PRINT"Intenten evitar chocar el uno con el
    otro y con las paredes."
40 INPUT"Como se llama el jugador de la izquierd
   a";nombrejug1$
50 PRINT"Use las teclas z y x para ir a izquierd
   ay derecha."
60 PRINT"Use las teclas d y c para subir y bajar
   ."
70 PRINT"Su caracter es: ";:PEN 3:PRINT CHR$(143
   )
80 PRINT:PEN 1
90 INPUT"Como se llama el jugador de la derecha"
   ;nombrejug2$
100 PRINT"Use las teclas , y . para ir a izquier
    day derecha."
110 PRINT"Use las teclas / y ; para subir y baja
    r."
120 PRINT"Su caracter es: ";:PEN 2:PRINT CHR$(14
    3)
130 PRINT:PEN 1
140 INPUT"Pulsen la tecla ENTER para empezar: ",
    empezar$
```

Ahora vamos a dibujar un campo de juego rectangular y establecer las posiciones de partida:

```
150 PAPER 1
160 PEN 0
170 CLS
180 pared$=CHR$(233)
190 expl$=CHR$(238)
200 izquierda=1
210 fondo=20
220 derecha=40
230 arriba=1
240 FOR cuenta=arriba TO fondo
250 LOCATE izquierda,cuenta
260 PRINT pared$;
270 LOCATE derecha,cuenta
280 PRINT pared$;
290 NEXT
300 LOCATE izquierda,arriba
310 PRINT STRING$(derecha-izquierda,pared$)
320 LOCATE izquierda,fondo
```

```
330 PRINT STRING$(derecha-izquierda,pared$)
340 jug1x=13
350 jug1y=7
360 jug1xmov=1
370 jug1ymov=0
380 jug1$=CHR$(143)
390 jug1choque=0
400 jug2x=26
410 jug2y=14
420 jug2xmov=-1
430 jug2ymov=0
440 jug2$=CHR$(143)
450 jug2choque=0
460 PEN 3
470 LOCATE jug1x,jug1y
480 PRINT jug1$;
490 PEN 2
500 LOCATE jug2x,jug2y
510 PRINT jug2$;
```

Finalmente, programamos el juego en sí, que sólo se interrumpe cuando un jugador choca con una pared o atraviesa el rastro del otro:

```
520 WHILE jug1choque=0 AND jug2choque=0
530 respuesta$=INKEY$
540 IF respuesta$="z" THEN jug1xmov=-1:jug1ymov=
    0
550 IF respuesta$="x" THEN jug1xmov=1:jug1ymov=0
560 IF respuesta$="d" THEN jug1xmov=0:jug1ymov=-
    1
570 IF respuesta$="c" THEN jug1xmov=0:jug1ymov=1
580 jug1x=jug1x+jug1xmov
590 jug1y=jug1y+jug1ymov
600 grafx=8+(jug1x-1)*16
610 grafy=8+(25-jug1y)*16
620 color=TEST(grafx,grafy)
630 LOCATE jug1x,jug1y
640 IF color<>1 THEN PEN 0:PRINT expl$:jug1choqu
    e=1 ELSE PEN 3:PRINT jug1$;
650 IF respuesta$="," THEN jug2xmov=-1:jug2ymov=
    0
660 IF respuesta$="." THEN jug2xmov=1:jug2ymov=0
670 IF respuesta$=";" THEN jug2xmov=0:jug2ymov=-
    1
680 IF respuesta$="/" THEN jug2xmov=0:jug2ymov=1
690 jug2x=jug2x+jug2xmov
```

```
700 jug2y=jug2y+jug2ymov
710 grafx=8+(jug2x-1)*16
720 grafy=8+(25-jug2y)*16
730 color=TEST(grafx,grafy)
740 LOCATE jug2x,jug2y
750 IF color<>1 THEN PEN 0:PRINT expl$:jug2choqu
    e=1 ELSE PEN 2:PRINT jug2$;
760 WEND
770 LOCATE 1,24
780 IF jug1choque=1 AND jug2choque=1 THEN PRINT"
    Fue un empate.":END
790 IF jug1choque=1 THEN PRINT "Gano ";nombrejug
    2$ ELSE PRINT "Gano ";nombrejug1$
800 PEN 1
810 PAPER 0
```

Para hacer más rápido el programa, sólo vamos a permitir el giro hacia la izquierda o la derecha, pues así sólo tenemos que comprobar dos letras por jugador, en lugar de cuatro:

```
540 IF respuesta$="z" OR respuesta$="x" THEN IF
    jug1xmov<>0 THEN jug1ymov=-jug1xmov:jug1xmov
    =0 ELSE jug1xmov=jug1ymov:jug1ymov=0
550 IF respuesta$="x" THEN jug1xmov=-jug1xmov:ju
    g1ymov=-jug1ymov
650 IF respuesta$="," OR respuesta$="." THEN IF
    jug2xmov<>0 THEN jug2ymov=-jug2xmov:jug2xmov
    =0 ELSE jug2xmov=jug2ymov:jug2ymov=0
660 IF respuesta$="." THEN jug2xmov=-jug2xmov:ju
    g2ymov=-jug2ymov
```

Estas líneas resultan muy complicadas para un ahorro de tiempo tan insignificante, pero usted debe tratar de entenderlas.

Otra forma de acelerar el programa sería no tener que convertir las coordenadas de texto en coordenadas gráficas. La solución, en la siguiente sección.

Ejercicios

1. Modifique el programa del laberinto para medir el tiempo que tarda el jugador en llegar al objetivo. Calcule un tanteo basado en el tiempo y en el nivel de habilidad.
2. Introduzca otra modificación en el programa del laberinto: dibuje algunos obstáculos de otro color para luego reducir el tanteo cada vez que la flecha choque con ellos.
3. Modifique el programa de los jugadores para que continúe hasta que uno de

de los jugadores haya ganado tres juegos. Escriba los nombres de los jugadores y el tanteo del rectángulo de juego.

4. Dificulte el programa de los dos jugadores diseñando un terreno de juego no rectangular. Diseñe dos caracteres nuevos para representar los dos jugadores.
5. Diseñe cuatro caracteres que representen un coche de carreras apuntando en cuatro direcciones. Escriba un programa que permita controlar por el teclado el movimiento del coche por una pista negra dibujada sobre fondo verde. Si el coche se sale de la pista, se desintegra. El programa será más completo y manejable si dedica una tecla a la opción "freno". Controle el tiempo y haga que el juego termine cuando el coche llegue a la línea de meta, la cual debe estar dibujada con un color diferente.

TEXTO EN LA POSICIÓN DEL CURSOR GRÁFICO

Esto de convertir coordenadas de texto en coordenadas gráficas es muy instructivo, pero en la práctica lo que demuestra es un defecto de enfoque en los programas anteriores. Disponemos de una resolución de 16Ø × 2ØØ puntos, y sin embargo sólo estamos aprovechado una resolución de 2Ø × 25, que es la que proporcionan las posiciones de texto. Incluso la resolución de texto en modo 2, que es de 8Ø caracteres por 25 líneas, es muy pobre comparada con la resolución de la pantalla gráfica. Lo que necesitamos es ser capaces de escribir caracteres en *cualquier* posición gráfica. De esa forma mejoraremos considerablemente la resolución y podremos olvidarnos de la conversión de coordenadas.

La instrucción que nos va a ayudar ahora es TAG. Aparte de otras ventajas, esta instrucción facilita el diseño de gráficas y diagramas, ya que nos permite escribir texto exactamente en la posición deseada. Por ejemplo, el siguiente programa dibuja un diagrama de barras de datos relativos a los doce meses del año y escribe la inicial del mes debajo de cada barra:

```
10 MODE 1
20 anchobarra=50
30 anchomes=16
40 anchorestante=anchobarra-anchomes
50 posicionmes=anchorestante/2
60 cerox=20
70 xmax=639
80 ceroy=50
90 ymax=350
100 MOVE cerox,ymax
110 DRAW cerox,ceroy,1
120 DRAW xmax,ceroy
130 TAG
140 FOR mes=1 TO 12
150 READ mes$,ventas
```

```
160 MOVE cerox+(mes-1)*anchobarra,ceroy
170 DRAW cerox+(mes-1)*anchobarra,ceroy+ventas,3
180 DRAW cerox+mes*anchobarra,ceroy+ventas
190 DRAW cerox+mes*anchobarra,ceroy
200 MOVE cerox+(mes-1)*anchobarra+posicionmes,ce
    roy-16
210 PRINT LEFT$(mes$,1);
220 NEXT
230 DATA Enero,97,Febrero,130,Marzo,141,Abril,15
    5,Mayo,210,Junio,276
240 DATA Julio,240,Agosto,223,Septiembre,112,Oct
    ubre,99.Noviembre.84.Diciembre,76
```

Obsérvese que los caracteres se escriben de tal forma que el punto superior izquierdo del carácter coincide con la posición del cursor gráfico. Hay que tener este hecho en cuenta antes de decidir dónde se va a escribir cada carácter. Esto explica los cálculos de las líneas 20 a 50, con los cuales se centra la letra bajo la barra correspondiente. Modifique la variable 'anchobarra' en la línea 20; si pone un valor mayor, observará que las iniciales siguen centradas, pero las barras son demasiado anchas y ya no caben todas en la pantalla.

La instrucción TAG se anula con TAGOFF. A partir del momento en que se ejecuta TAGOFF, la instrucción vuelve a enviar los textos a la posición en que se encontraba el cursor de texto antes de ejecutar TAG. El modo TAG se cancela automáticamente cuando el programa termina o es interrumpido.

No por el hecho de escribir con TAG nos vemos libres de las limitaciones que impone la resolución gráfica. Por ejemplo, el desplazamiento en vertical tiene que ser de al menos dos puntos; si sólo nos movemos un punto, el ordenador vuelve a escribir el carácter en la misma posición. El desplazamiento mínimo en horizontal depende del modo de pantalla. En la figura 14 se muestra cuál es ese desplazamiento mínimo para los tres modos. El siguiente programa ilustra lo que decíamos:

```
10 MODE 1
20 TAG
30 x=300
40 y=200
41 MOVE x,y:PRINT CHR$(249);
50 respuesta$=""
60 WHILE respuesta$<>"n"
70 respuesta$=INKEY$
80 IF respuesta$="z" THEN MOVE x,y:PRINT" ";:x=x
   -1:MOVE x,y:PRINT CHR$(249);
90 IF respuesta$="x" THEN MOVE x,y:PRINT" ";:x=x
   +1:MOVE x,y:PRINT CHR$(249);
100 WEND
```

Observe que hay que pulsar cada tecla al menos dos veces para que se produzca algún movimiento. La situación es aun peor si se ejecuta el programa en modo Ø, porque entonces hay que pulsar las teclas cuatro veces para conseguir un movimiento horizontal.

ORIGIN y movimientos relativos

El programa que dibuja los diagramas de barras se puede simplificar aun más si se dibuja cada barra después de modificar el origen de las coordenadas gráficas. Hemos visto antes que podemos definir ventanas de texto, y que cada una de ellas tiene su sistema de coordenadas propio, con el origen en el extremo superior izquierdo. De manera análoga podemos desplazar el sistema de coordenadas gráficas eligiendo qué punto queremos que tenga las coordenadas (Ø, Ø):

```
10 MODE 1
20 anchobarra=50
30 anchomes=16
40 anchorestante=anchobarra-anchomes
50 posicionmes=anchorestante/2
60 cerox=20
70 xmax=639
80 ceroy=50
90 ymax=350
100 MOVE cerox,ymax
110 DRAW cerox,ceroy,1
120 DRAW xmax,ceroy
121 ORIGIN cerox,ceroy
125 MOVE 0,0
130 TAG
140 FOR mes=1 TO 12
150 READ mes$,ventas
170 DRAW 0,ventas,3
180 DRAW anchobarra,ventas
190 DRAW anchobarra,0
200 MOVE posicionmes,-16
210 PRINT LEFT$(mes$,1);
215 cerox=cerox+anchobarra
216 ORIGIN cerox,ceroy
220 NEXT
230 DATA Enero,97,Febrero,130,Marzo,141,Abril,15
    5,Mayo,210,Junio,276
240 DATA Julio,240,Agosto,223,Septiembre,112,Oct
    ubre,99,Noviembre,84,Diciembre,76
```

Como se puede observar, los cálculos previos al dibujo de cada barra son ahora más sencillos. El bucle de la línea 140 desplaza el origen de coordenadas hasta el sitio donde se va a dibujar cada barra. Las órdenes DRAW son también más sencillas.

El interés principal de la orden ORIGIN está en que permite repetir una figura determinada en diversos lugares de la pantalla.

A menudo se observa que un dibujo se simplifica cuando lo programamos en coordenadas *relativas* en lugar de en coordenadas *absolutas*. Observe el siguiente programa, que dibuja un rectángulo:

```
10 MODE 1
20 MOVE 0,0
30 DRAW 200,0
40 DRAW 200,100
50 DRAW 0,100
60 DRAW 0,0
```

En este programa las coordenadas son absolutas. Así, en la línea 30 las coordenadas son las del punto (200, 0); por muchas veces que ejecutemos el programa, el dibujo siempre será el mismo. A pesar de lo sencilla que es esta figura, es un trabajo tedioso tener que calcular las coordenadas de los vértices cada vez que queramos dibujarla en otra posición de la pantalla. Sería muy conveniente poder describir los movimientos refiriéndolos a la posición actual del cursor gráfico. Los movimientos relativos necesarios para dibujar el rectángulo serían los que se muestran en la figura 31.

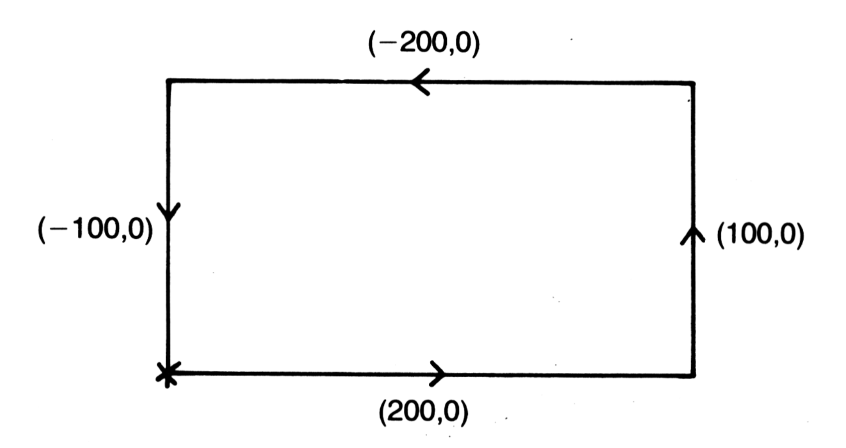


Figura 31. Desplazamientos necesarios para dibujar un rectángulo, referido a la posición actual del cursor gráfico.

Como era de esperar, el BASIC del Amstrad dispone de las instrucciones que resuelven este problema. Se trata de MOVER, DRAWR y PLOTR, con las que se realizan movimientos relativos sin tener que cambiar el origen de coordenadas:

```
10 MODE 1
15 INPUT "Coordenadas del vertice inferior
     izquierdo del rectangulo";x,y
20 MOVE x,y
30 DRAWR 200,0
40 DRAWR 0,100
50 DRAWR -200,0
60 DRAWR 0,-100
```

Las órdenes DRAWR de las líneas 30 a 60 especifican desplazamientos referidos a la última posición visitada. La línea 30 dibuja una recta que va desde el último punto visitado hasta un punto que se encuentra a 200 puntos a su derecha y en la misma horizontal. La línea 60 dibuja una recta que va desde el último punto visitado hasta un punto que está en la misma vertical y a 100 puntos por encima de él.

Incluyendo un factor de escala podemos dibujar una serie de rectángulos semejantes:

```
10 MODE 1
15 INPUT "Coordenadas del vertice inferior
     izquierdo del rectangulo";x,y
20 MOVE x,y
25 INPUT "Escala";escala
30 DRAWR 200*escala,0
40 DRAWR 0,100*escala
50 DRAWR -200*escala,0
60 DRAWR 0,-100*escala
```

Introduciendo un valor de 'escala' de 2, se dibuja un rectángulo de tamaño doble que el programa anterior. Con un factor de escala de 0.1 el rectángulo se reduce a la décima parte. Aunque en estos programas no hayamos tenido ocasión de hacerlas intervenir, las instrucciones MOVER y PLOTR funcionan de forma similar. Cuando hay que dibujar varias figuras iguales o compuestas por elementos que se repiten, los programas se simplifican mucho si se utiliza ORIGIN y los desplazamientos relativos.

Ejercicios

1. Vuelva a escribir el programa del laberinto haciendo que la flecha se mueva recorriendo posiciones gráficas en lugar de posiciones de texto.
2. Modifique el programa del diagrama de barras para que escriba un título debajo del diagrama. Rotule el eje para que se puede leer aproximadamente la longitud de las barras.
3. Escriba un programa que simule 200 lanzamientos de dos dados y dibuje un diagrama de barras en el que se muestre la distribución de los resultados. Rotule los ejes horizontal y vertical.
4. Escriba un programa que dibuje la siguiente figura con MOVER y DRAWR:

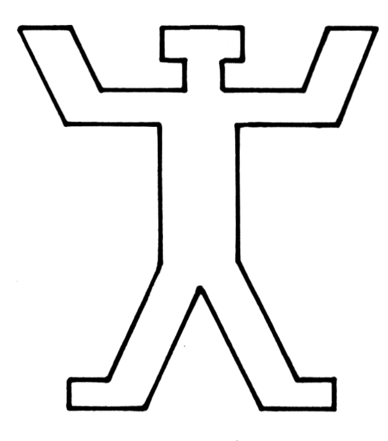


Figura 32.

5. Amplie el programa del ejercicio anterior para que dibuje una "pirámide humana", cada "piso" con un color diferente.

Plumas y tinteros

Hasta ahora sólo hemos visto parte de los colores que el Amstrad puede generar. Hay un máximo de 16 plumas (en modo Ø), y sin embargo los colores que relacionábamos en la figura 18 son 27. El Amstrad nos permite decidir en cuál de los 27 tinteros queremos cargar cada una de las 16 plumas. De esta forma podemos seleccionar una combinación de colores, cuya variedad dependerá del modo de pantalla.

El máximo número de colores que puede haber en la pantalla simultáneamente sigue estando limitado. Por ejemplo, en modo 2, aunque decidamos escribir con caracteres rojos sobre fondo blanco, éstos serán los *dos únicos* colores que podremos tener en la pantalla en un momento dado. Los límites son, pues 2 colores en modo 2, 4 en modo 1 y 16 en modo Ø.

Cuando encendemos o reinicializamos la máquina, se pone automáticamente en modo 1 y selecciona el papel número Ø (PAPER Ø, que inicialmente es azul porque está cargado con tinta número 1) y la pluma número Ø (PEN 1, inicialmente amarilla, tinta número 24). Reinicialice el ordenador y escriba:

```
INK 1,6
```

El texto que había en la pantalla ha cambiado instantáneamente de amarillo a rojo intenso. La orden INK va seguida de dos números: el primero es el número del papel y de la pluma cuyo color se va a cambiar; el segundo es el número de la tinta con que se va a cargar ese papel y esa pluma.

Así pues, la orden INK 1,6 ha hecho que el papel número 1 y la pluma número 1 se carguen con tinta de color 6, que es el rojo intenso. En el momento de ejecutar la orden, cambia de color todo lo que antes se hubiera escrito con la pluma 1 o sobre

fondo de papel 1. Para volver de color azul intenso el texto que tiene en la pantalla, escriba

```
INK 1,2
```

Todo lo que antes era rojo ahora es azul. ¿Cómo devolver el color normal al texto? Quizá sepa usted la respuesta:

```
INK 1,24
```

Inicialmente, la pluma 1 está cargada con tinta (INK) 24 en todos los modos, como puede comprobar consultando la figura 2Ø.

El color del fondo se cambia con la misma facilidad. En este momento el ordenador está utilizando el papel número Ø. Como no se lo hemos cambiado, su color sigue siendo azul. Para hacerlo blanco escriba

```
INK Ø,26
```

El texto es difícil de leer. Pruebe con

```
INK Ø,6
```

o con

```
INK Ø,Ø
```

El papel Ø es inicialmente azul (tinta número 1) en todos los modos. Esta vez vamos a dejar que devuelva usted todos los colores a la normalidad sin ayuda.

No es necesario haber utilizado previamente un papel o una pluma para cambiarles la tinta. Reinicialice el ordenador y escriba:

```
INK 3,Ø
```

No parece que haya ocurrido nada. Si ahora elegimos la pluma 3 en modo 1, comprobaremos que está cargada con tinta negra (número Ø), no con la tinta roja (número 6) que podemos ver en la figura 2Ø. Escriba

```
PEN 3
```

y verá cómo todo el texto siguiente aparece en negro. Escriba

```
INK 3,6
```

Todo lo que antes era negro ahora se ha vuelto rojo (tinta número 6). La pluma 3, que es la que estamos utilizando en este momento, empieza a escribir con tinta roja. Esta situación permanece así hasta que se cambie de modo de pantalla.

Compruébelo.

Se puede cargar un número de pluma o de papel con una mezcla de dos colores:

```
INK 1,3,26
```

El texto escrito con la pluma 1 alterna ahora entre los colores 3, rojo, y 26, blanco.

La ventaja obvia de INK es que nos permite seleccionar una combinación cualquiera de los 27 colores disponibles, aunque con la limitación del número total de colores simultáneos, el cual depende del modo de pantalla. Incluso el modo 2, con su modesta variedad, se puede animar eligiendo color blanco para el fondo y rojo para el texto.

El siguiente programa genera las 749 combinaciones de color posibles en modo 2:

```
10 MODE 2
20 FOR x=0 TO 27
30 CLS
40 INK 0,x
50 FOR y=0 TO 27
60 IF x<>y THEN INK 1,y:PRINT"ink ";y
70 respuesta$=""
80 WHILE respuesta$=""
90 respuesta$=INKEY$
100 WEND
110 NEXT
120 NEXT
```

Una ventaja no tan evidente de este sistema de selección de color es que se puede elegir una pluma que esté cargada con tinta del mismo color que el fondo para escribir mensajes o dibujar figuras y luego hacerlos aparecer instantáneamente con una simple instrucción INK:

```
10 MODE 0
20 REM dibujar monigotes con plumas diferentes
30 FOR x=1 TO 17 STEP 4
40 FOR y=1 TO 20 STEP 4
50 PEN 1
60 REM ponerlos del color del fondo
70 INK 1,1
80 LOCATE x,y
90 PRINT CHR$(248);
100 NEXT
110 NEXT
120 PEN 5
130 INK 5,24
140 LOCATE 1,22
150 PRINT "Pulse cualquier     tecla para ver lo
    s  monigotes";
```

```
160 REM esperar pulsacion de tecla
170 respuesta$=""
180 WHILE respuesta$=""
190 respuesta$=INKEY$
200 WEND
210 INK 1,24
220 GOTO 220
```

También podemos cargar varias plumas con el color del fondo y luego cambiarles la tinta selectivamente:

```
10 MODE 0
11 REM dibujar monigotes con diferentes colores
   de pluma
20 FOR x=1 TO 17 STEP 4
30 FOR y=1 TO 20 STEP 4
40 PEN 1
41 REM ponerlos del color del fondo
50 INK 1,1
60 LOCATE x,y
70 PRINT CHR$(248);
80 PEN 2
81 REM ponerlos del color del fondo
90 INK 2,1
100 PRINT CHR$(249);
110 PEN 3
111 REM ponerlos del color del fondo
120 INK 3,1
130 PRINT CHR$(250);
140 PEN 4
141 REM ponerlos del color del fondo
150 INK 4,1
160 PRINT CHR$(251);
170 NEXT
180 NEXT
190 PEN 5
200 INK 5,24
210 LOCATE 1,22
220 PRINT "Pulse cualquier     tecla para mover
    losmonigotes";
221 REM ciclo de colores
230 FOR color=0 TO 50
231 REM esperar pulsacion de tecla
240 respuesta$=""
250 WHILE respuesta$=""
260 respuesta$=INKEY$
```

```
27Ø WEND
271 REM calcular color para fondo
28Ø nover=color MOD 4
29Ø IF nover=Ø THEN nover=4
3ØØ INK nover,1
3Ø1 REM calcular color para figuras
31Ø ver=(color+1) MOD 4
32Ø IF ver=Ø THEN ver=4
33Ø INK ver,24
34Ø NEXT
341 REM volver pen 1 a normal
35Ø PEN 1
36Ø INK 1,24
```

Con la línea 23Ø recorremos los colores de tinta del Ø al 5Ø. Como sólo nos interesan los colores del 1 al 4, con las líneas 28Ø a 31Ø obtenemos números de este margen. La función MOD da el resto de la división; por ejemplo, 17 MOD 4 da 1, porque 1 es el resto que se obtiene al dividir 17 por 4. Nos queda un pequeño problema, y es que los números producidos por MOD van del Ø al 3, no del 1 al 4. Por eso hemos incluido las líneas 29Ø y 32Ø, para convertir el Ø en 4. El efecto global es que, en cada pasada por el bucle, cargamos una pluma con tinta del color del fondo (1) y la siguiente con tinta del color del texto (24), dando así una ilusión de movimiento de los monigotes, cuando en realidad lo que estamos haciendo es apagarlos y encenderlos como si fueran bombillas.

Ejercicios

1. Escriba un programa que cambie los colores de una tinta de modo que un mensaje aparezca, en modo 1, en caracteres morados, negros y verdes sobre fondo blanco.
2. Añada unas líneas al programa del laberinto para que la pantalla se dibuje con el color del fondo y luego aparezca súbitamente con una orden INK.
3. Dibuje una casa sobre un cielo azul, con hierba verde delante. Al tocar una tecla, los colores deben cambiar la escena para que parezca que es de noche y sale luz por las ventanas.
4. Provoque un incendio en la casa dibujando algo a su alrededor en colores parpadeantes adecuados.

9

Planificación de un programa

EN QUE PUEDE AYUDARNOS EL AMSTRAD

En los últimos capítulos hemos avanzado mucho. Los programas han ido siendo cada vez más largos y más complejos. A estas alturas el lector ya conocerá a fondo los métodos de edición del Amstrad, pero hay muchas otras funciones del ordenador que pueden ayudarle en la elaboración de programas. Una de las cosas que vamos a estudiar en este capítulo es cómo utilizar funciones del ordenador para detectar los errores que inevitablemente se deslizan al escribir los programas.

No obstante, el objetivo principal de este capítulo es dejar bien claro lo importante que es la planificación en todo trabajo de programación. Le daremos unas sugerencias sobre cómo evitar los problemas que suelen dar los programas mal organizados. Como instrumento para exponer estas ideas, desarrollaremos un programa de juego, desde su concepción hasta su versión final libre de errores.

Antes de nada, vamos a hacer unos preparativos para trabajar con mayor comodidad.

Las teclas de función

El lector habrá observado que hay determinadas órdenes que se utilizan con extraordinaria frecuencia; por ejemplo, LIST y RUN. A partir de ahora podrá listar y ejecutar los programas con sólo pulsar una tecla. (Cargue o escriba un pequeño programa para tener algo que listar y ejecutar.)

Vimos en el capítulo 6 que todo carácter tiene asociado un código ASCII. Los códigos que van del Ø a 31 tienen significados especiales. Los del margen 32 a 127 representan las letras mayúsculas y minúsculas, las cifras, los signos de puntuación y otros caracteres de uso corriente. Los del margen 128 a 159 representan diversos caracteres gráficos, pero también se los puede asociar a cadenas literales, que entonces pueden ser generadas al pulsar la tecla correspondiente; son los *códigos expansibles*.

En el momento de inicializar o encender la máquina, los códigos expansibles del 128 al 14Ø son generados por el teclado numérico y tienen asignadas *las cadenas de expansión* que se muestran en la figura 33. Los restantes códigos expansibles, del 141 al 159, no están asignados inicialmente a ninguna tecla.

Código expansible	Asignación inicial	
	Tecla	Cadena de expansión
128	Ø	"Ø"
129	1	"1"
13Ø	2	"2"
131	3	"3"
132	4	"4"
133	5	"5"
134	6	"6"
135	7	"7"
136	8	"8"
137	9	"9"
138	.	"."
139	⟨ENTER⟩	CHR$(13)
14Ø	⟨CTRL⟩⟨ENTER⟩	"RUN"+CHR$(13)

Figura 33. Códigos expansibles: posiciones y valores implícitos.

Mediante la instrucción KEY ("tecla") podemos modificar la cadena de expansión asignada a los caracteres expansibles. Por ejemplo,

```
KEY 128,"Hola, amigo lector"
```

Esta orden hace que, en lo sucesivo, cada vez que pulsemos la tecla del 'Ø' en el teclado numérico, obtengamos la frase 'Hola, amigo lector'. Las teclas del teclado numérico se denominan *teclas de función* por la facilidad con que pueden ser programadas. No obstante, *todas las demás teclas son programables*, como veremos más adelante.

Una aplicación más útil de KEY sería

```
KEY 128,"LIST"
```

Obsérvese que el número del código debe ir seguido de una coma y que la cadena de expansión se debe especificar siguiendo las reglas de las cadenas literales. Si ahora pulsa la tecla del 'Ø', el ordenador escribe la palabra LIST, pero el cursor se queda a la derecha de ella, esperando que usted pulse ⟨ENTER⟩. Esto está bien para algunas funciones, pero en ocasiones preferimos que el propio ordenador "pulse" ⟨ENTER⟩.

Pues bien, la acción de pulsar ⟨ENTER⟩ se especifica mediante uno de los códigos del Ø al 31; concretamente, con el 13. Escriba:

```
KEY 128,"LIST"+CHR$(13)
```

y pulse la tecla del 'Ø'.

No es mala idea incluir 〈ENTER〉 al principio de la cadena de expansión. De lo contrario, si por ejemplo acabamos de escribir una línea del programa y todavía no hemos pulsado 〈ENTER〉, al pulsar la tecla del 'Ø' lo que hacemos es equivalente a escribir LIST al final de la línea y luego pulsar 〈ENTER〉. En cambio, si la definición es

```
KEY 128,CHR$(13)+"LIST"+CHR$(13)
```

al pulsar la tecla del 'Ø' el ordenador da por terminada la línea pendiente y luego lista el programa.

En la cadena de expansión se puede incluir cualquier palabra clave de BASIC. Por ejemplo, es conveniente volver al modo 1 antes de listar el programa:

```
KEY 128,CHR$(13)+"MODE 1:LIST"+CHR$(13)
```

Como los números siguen estando disponibles en la primera fila del teclado principal, nada impide que asignemos cadenas de expansión a todas las teclas del teclado numérico. Quizá le interese grabar el siguiente conjunto de definiciones para utilizarlo sistemáticamente en el futuro:

```
10 KEY 128,CHR$(13)+"mode 1:list"+CHR$(13)
20 KEY 129,CHR$(13)+"run"+CHR$(13)
30 KEY 130,CHR$(13)+"save"
40 KEY 131,CHR$(13)+"load"
50 KEY 138,CHR$(13)+"auto"
60 KEY 139,CHR$(13)+"cls"+CHR$(13)
70 KEY 132,"while"
80 KEY 133,"wend"
90 KEY 134,"for"
100 KEY 135,"next"
110 KEY 136,"read"
120 KEY 137,"data"
```

o modifíquelo según su gusto personal o sus necesidades concretas.

La gran ventaja de que las teclas sean definibles por programa es que se puede tener grabados varios programas, cada uno con un juego de definiciones distintas. En cada sesión de trabajo se carga y ejecuta el programa deseado; cuando se borra el programa, no se pierde la definición de las teclas. Por ejemplo, cuando se está desarrollando programas gráficos, el siguiente conjunto de definiciones puede ser útil:

```
10 KEY 128,CHR$(13)+"mode 1:list"+CHR$(13)
20 KEY 129,CHR$(13)+"run"+CHR$(13)
30 KEY 130,CHR$(13)+"save"
40 KEY 131,CHR$(13)+"load"
```

```
50 KEY 138,CHR$(13)+"auto"
60 KEY 139,CHR$(13)+"cls"+CHR$(13)
70 KEY 132,"move"
80 KEY 133,"draw"
90 KEY 134,"plot"
100 KEY 135,"pen"
110 KEY 136,"window"
120 KEY 137,"data"
```

La única forma de saber qué cadena de expansión tiene asignada cada tecla en un momento dado es pulsar las teclas. Este pequeño inconveniente no debe desanimarle; si utiliza siempre el mismo juego de definiciones, puede pegar unas etiquetas autoadhesivas en las teclas; si no, confíe en su memoria.

La definición de teclas no tiene más límite que el siguiente: el máximo número de caracteres que pueden contener las cadenas de expansión es 12Ø.

Estos 12Ø caracteres se pueden repartir como se desee entre todas las cadenas. No es probable que esta limitación tenga ninguna importancia, a menos que su trabajo se salga de lo normal.

Definición de otras teclas

Como hemos dicho, todas las teclas son definibles, aunque las del teclado principal requieren una instrucción más. Ilustraremos el procedimiento con un ejemplo. Vamos a redefinir la tecla de la 'A' para que genere, en minúsculas, la 'Z' mayúscula:

```
KEY DEF 69,1,90
```

El primer parámetro de KEY DEF es el *número de tecla*. En el apéndice III del Manual del Usuario se da un mapa con los números de todas las teclas. El segundo parámetro puede ser Ø o 1; si es 1, la tecla será de repetición; si es Ø, no lo será. El tercer parámetro es el código ASCII que queremos que la tecla genere en minúsculas; en este caso hemos puesto el 9Ø, que representa la 'Z' mayúscula, pero *también podríamos haber puesto cualquier código expansible*. Los parámetros cuarto y quinto, omitidos en este caso, son los códigos ASCII que la tecla debe producir en mayúsculas y combinadas con ⟨CTRL⟩, respectivamente.

Para devolver la tecla a su situación normal escriba

```
KEY DEF 69,1,97
```

donde 97 es el código ASCII de la 'a' minúscula.

Este ejemplo no revela ninguna utilidad. Sin embargo, hay ciertas aplicaciones, como en proceso de textos, en las que es conveniente poder redefinir teclas para adaptar el teclado al idioma (acentos, tilde, etc.) Por otra parte, seguimos disponiendo de los códigos expansibles, algunos de los cuales (del 141 al 159) no hemos tenido ocasión de utilizar. Para servirnos de ellos necesitamos KEY DEF y KEY. Vamos a tomar uno de ellos, el 141, y a asignarlo a la tecla de la libra esterlina, '£':

```
KEY DEF 24,0,141
```

A partir de este momento la tecla '£' produce el código 141. Sin embargo, al pulsarla no vemos que aparezca nada en la pantalla. Es natural, puesto que todavía no hemos asignado ninguna cadena de expansión al código 141. Escriba

```
KEY 141,"read"
```

y pulse la tecla de la '£'. Hemos estableceido así una conexión indirecta entre el teclado y los códigos expansibles.

El método para definir una tecla que no esté en el teclado numérico es el siguiente:

1) Elegir una tecla que no se utilice habitualmente; averiguar su número consultando el apéndice III del Manual del Usuario.
2) Redefinir la tecla con KEY DEF para que genere uno de los caracteres expansibles (de 120 al 159).
3) Asignar a ese caracter, con KEY, la cadena de expansión deseada.

Tambien se puede, con KEY DEF, modificar una tecla del teclado numérico para que produzca un código expansible distinto del inicialmente asignado o cualquier otro código ASCII. Por ejemplo, KEY DEF 6,1,65,66,130 redefine la tecla ⟨ENTER⟩ pequeña para que genere la letra 'A', la letra 'B' y el código expansible 130 en minúsculas, en mayúsculas y con ⟨CTRL⟩, respectivamente.

El siguiente programa redefine cinco de las teclas que se utilizan con menor frecuencia:

```
10 KEY 128,CHR$(13)+"mode 1:list"+CHR$(13)
20 KEY 129,CHR$(13)+"run"+CHR$(13)
30 KEY 130,CHR$(13)+"save"
40 KEY 131,CHR$(13)+"load"
50 KEY 138,CHR$(13)+"auto"
60 KEY 139,CHR$(13)+"cls"+CHR$(13)
70 KEY 132,"move"
80 KEY 133,"draw"
90 KEY 134,"plot"
100 KEY 135,"pen"
110 KEY 136,"window"
120 KEY 137,"data"
121 REM tecla de libra esterlina
130 KEY DEF 24,0,141
140 KEY 141,"read"
141 REM tecla de la 'arroba'
150 KEY DEF 26,0,142
160 KEY 142,"while"
161 REM tecla de abrir corchete
170 KEY DEF 17,0,143
180 KEY 143,"wend"
```

```
181 REM tecla de cerrar corchete
190 KEY DEF 19,0,144
200 KEY 144,"for"
201 REM tecla de la barra inclinada hacia la izq
    uierda
210 KEY DEF 22,0,145
220 KEY 145,"next"
```

Otra tecla que se podría redefinir es la ⟨TAB⟩, pero con esto casi se agota el repertorio. La forma de ampliar las posibilidades de redefinición de teclas es una que ya hemos mencionado: dar definiciones distintas para minúsculas, mayúsculas y ⟨CTRL⟩. Por ejemplo,

```
KEY DEF 22,0,145,146,147
KEY 145,"if"
KEY 146,"then"
KEY 147,"else"
```

redefine la tecla '\' para que produzca tres palabras diferentes en minúsculas, mayúsculas y combinada con ⟨CTRL⟩.

Ejercicios

1. Diseñe usted una redefinición de las teclas y grabe el programa para su utilización posterior. Algunas funciones que se pueden incluir en las cadenas de expansión son las siguientes: devolver los números de pluma y de papel, así como sus colores, a la situación habitual; asignar 'KEY' y 'KEY DEF' a alguna tecla para facilitar las redefiniciones; asignar a diversas teclas los nombres de variables más frecuentemente utilizados, tales como 'continuar', 'cuenta', 'numeropluma', etc.

ERRORES Y DEPURACIÓN

Al principio de este capítulo mencionábamos la cuestión de los errores en los programas. Contra lo que el profano puede creer, no hay ningún programa que funcione bien a la primera, salvo los más triviales. Casi todos los programas son el resultado de una fase de planificación seguida de otra fase en la que el programador se dedica a resolver problemas que no había previsto antes. Si la planificación no ha sido cuidadosa, en ocasiones los cambios necesarios son tantos que equivalen a escribir el programa de nuevo.

No le sorprendan ni desanimen los errores, en particular si es usted nuevo en la afición a la informática. El ordenador dispone de recursos que ayudan a detectar errores; utilizándolos le será mucho más fácil depurar los programas. Veamos un pequeño programa que ilustra el funcionamiento de esos recursos:

```
10 ON ERROR GOTO 100
20 pront"Hola"
30 END
100 PRINT"Hay un error en la linea";ERL
110 PRINT"El numero de error es";ERR
120 END
```

En la línea 20 hemos dejado deliberadamente un error, 'pront' en lugar de 'print'. La línea 10 significa: "cuando encuentres un error, salta a la línea 100". Y, en efecto, cuando el programa llega a la línea 20, detecta un error y salta a la línea 100. Ésta escribe el número de línea donde ha detectado el error; la línea 110 escribe el número del error, en este caso el 2. En el apéndice VIII del Manual del Usuario se da una lista de los números de los errores junto con su descripción. El error 2 es un *syntax error* (error de sintaxis), lo que significa que hemos escrito algo sin ajustarnos a las reglas "gramaticales" de BASIC.

Gracias a la línea 110, sabemos dónde ha ocurrido el error y así podemos editar la línea que lo contiene. Curiosamente, conseguimos mayor información omitiendo la línea 10 que con ella; si lo dejamos solo, el Amstrad describe el error en lugar de limitarse a indicar el número. Los demás ordenadores no suelen ser tan explícitos; en ellos ON ERROR es más importante. No obstante, en algunas situaciones, ON ERROR puede ser interesante también en el Amstrad, como demuestra el siguiente programa:

```
10 ON ERROR GOTO 1000
20 PEN 0
30 papper 1
40 END
1000 MODE 1
1010 INK 1,24
1020 INK 0,1
1030 PEN 1
1040 PAPER 0
1050 PRINT"Hay un error en la linea";ERL
1060 PRINT"El numero de error es";ERR
1070 END
```

El ordenador detecta el error de la línea 30 después de seleccionar la pluma 0 en la línea ; si no fuera por la línea 10, el Amstrad escribiría el mensaje de error con esa pluma, que tienen el mismo color que el papel; es decir, el mensaje sería invisible. Pero ON ERROR reconduce el programa a la línea 1000; en las líneas 1010 y siguientes restauramos los colores normales antes de escribir los mensajes en las líneas 1050 y 1060.

Todos los errores que hemos estudiado hasta ahora han sido errores de sintaxis, pero hay muchos otros que pueden presentarse en los programas. Si escribimos

numero="Hola"

provocamos un error de 'Type mismatch' (''incongruencia de tipos''); hemos tratado de asignar una cadena literal a una variable numérica. Estos errores, y muchos otros, pueden ser detectados porque transguedan las reglas que están grabadas en la ROM; y por ser detectables podemos corregirlos. Mucho más graves son los *errores lógicos* cometidos por el programador en la concepción del programa.

Un error lógico es aquél que hace que el programa no funcione como se espera de él. Por ejemplo, el siguiente:

```
10 MODE 1
20 numero1=5
30 numero2=7
40 PRINT numero1;"mas";numero2;"es igual a";nume
   ro1*numero2
50 END
```

El programa funciona a la perfección en el sentido de que no viola ninguna regla de BASIC. Sin embargo, no da el resultado correcto porque el programador ha cometido un error lógico en la línea 40: ha multiplicado los números en lugar de sumarlos. En este caso el error es obvio porque la operación es muy sencilla y el resultado aparece inmediatamente en la pantalla. Pero si el resultado se utilizara para realizar un cálculo posterior, y el de éste para otro, etc., el error ya no sería tan fácil de detectar.

Un error lógico muy frecuente se produce cuando se abre un bucle infinito:

```
10 ON ERROR GOTO 1000
20 continuar=0
30 WHILE continuar=0
40 WEND
50 END
1000 MODE 1
1010 INK 1,24
1020 INK 0,1
1030 PEN 1
1040 PAPER 0
1050 PRINT"Hay un error en la linea";ERL
1060 PRINT"El numero de error es";ERR
1070 END
```

Afortunadamente, este bucle se puede interrumpir pulsando dos veces ⟨ESC⟩, ya que esta interrupción es prioritaria con respecto a ON ERROR. Pero si hubiéramos puesto las líneas

```
22 PEN 2
23 INK 2,1
```

antes de entrar en el bucle, al pulsar ⟨ESC⟩ seguiríamos sin saber qué ocurre porque los textos serían invisibles. Esto se remedia con la siguiente línea:

```
10 ON BREAK GOSUB 1000
```

(*on break* significa "en caso de interrupción".) La línea 10 indica al ordenador qué debe hacer si tratamos de interrumpir el programa pulsando la tecla ⟨ESC⟩; en concreto, ir a la línea 1000 y ejecutar las instrucciones que allí encuentre. Ahora las líneas 1050 y 1060 ya no sirven para nada; en cambio, las líneas 1000 a 1040 restauran las condiciones normales de la pantalla, y esto es lo que necesitamos.

¿Por qué hemos puesto GOSUB en la línea 10 y no GOTO? Enseguida lo explicaremos.

Otra instrucción aplicable a la detección de errores es STOP:

```
10 MODE 1
20 FOR x=1 TO 100
30 IF x MOD 5=0 THEN STOP
40 NEXT
```

STOP nos permite detener momentáneamente el programa, examinar las variables de interés y reanudar la ejecución. Si ejecutamos el programa anterior, éste se detiene y emite el mensaje 'Break in 30' ("interrupción en 30"). Escriba '?x' y el ordenador le dirá cuál es el valor de la variable 'x' en este momento: 5. Si ahora escribe CONT, el programa continúa, a partir de la situación en que se detuvo, hasta que 'x' valga 10, y así sucesivamente. En un programa más largo, cuando no se está seguro de cómo evoluciona, se insertan instrucciones STOP en puntos estratégicos para examinar el valor de las variables sospechosas. También se pueden definir unas cuantas teclas de función para que escriban el valor de las variables de interes.

ORGANIZACIÓN DEL PROGRAMA

En la sección anterior vimos por primera vez una instrucción GOSUB, pero no la utilizamos correctamente. Lo que hace GOSUB es decirle al ordenador que vaya a la línea especificada y continúe a partir de ella hasta que encuentre una instrucción RETURN:

```
10 MODE 1
20 GOSUB 100
30 PRINT"Fuera de la subrutina."
40 END
100 FOR cuenta=1 TO 10
110 PRINT"Contador dentro de la subrutina:";cuen
    ta
120 NEXT
130 RETURN
```

La línea 20 reconduce el programa hacia la *subrutina* que empieza en la línea 100. El ordenador obedece las instrucciones de las líneas 100 a 120; al llegar a la 130, "retorna" a la línea siguiente a aquélla que invocó la subrutina, esto es, a la 30.

La línea 40 es muy importante, pues impide que el programa llegue a la subrutina por la vía indebida.

GOSUB es la primera instrucción que hemos conocido que altera el orden normal de ejecución de los programas, que es de menor a mayor número de línea.

En los últimos capítulos los programas se nos han ido complicando. En cuanto un programa tiene más de unas 10 líneas, lo más conveniente es desarrollarlo por secciones. Al disponer de las subrutinas, podemos dividir el programa en una serie de subprogramas, cada uno de los cuales realice una tarea específica, sencilla, y por consiguiente fácil de programar. Cada subrutina se prueba y depura hasta que funcione correctamente; luego se reúnen todas las subrutinas para formar el programa global.

Quizá no se entiendan bien estas ideas mientras no las ilustremos con un ejemplo, así que vamos a escribir un programa de juego. Se trata de un juego muy popular, del que hay versiones para todos los ordenadores.

Planificación de un programa

El juego consiste en que el jugador controla un avión que "vuela" de izquierda a derecha por la pantalla. Cada vez que el avión llega al borde derecho, reaparece por el izquierdo una línea más abajo. El avión está volando sobre una ciudad formada por edificios de alturas diferentes. Cada vez que el jugador pulsa la tecla 'b', el avión suelta una bomba; si cae sobre un edificio, lo destruye parcialmente por arriba; si no, cae sobre la hierba, donde explosiona. Mientras una bomba está cayendo no puede caer otra. El objetivo del juego es hacer aterrizar el avión, para lo cual hay que destruir todos los edificios a tiempo. Si esto no se consigue, el avión en algún momento se estrellará contra algún edificio, con lo que terminará el juego.

Desde el punto de vista del programador, podemos subdividir el problema en varias secciones:

1) Instrucciones para el jugador, nivel de juego, etc.
2) Inicialización de las variables que se van a utilizar, dibujo de la pantalla con los edificios y con el avión en la posición de partida.
3) Rutina de juego.
4) Comentarios sobre el resultado del juego, tanteo, restauración de los colores a la situación normal, etc.

Esta subdivisión no es la única posible. Por ejemplo, la sección 2 se podría subdividir a su vez en dos partes. Pero en este momento no nos preocupan los detalles del programa, sino su aspecto general.

Empecemos a escribir el esqueleto del programa:

```
5 ON ERROR GOTO 70
10 MODE 1
```

```
11 REM instrucciones
20 GOSUB 1000
21 REM preparar posicion de comienzo para el jue
   go
30 GOSUB 2000
31 REM jugar
40 GOSUB 3000
41 REM comentarios y marcador
50 GOSUB 4000
60 END
70 INK 0,1
80 INK 1,24
90 INK 2,20
100 INK 3,6
110 PEN 1
120 PAPER 0
130 PRINT"Error en la linea";ERL
140 PRINT"Numero de error:";ERR
150 END
1000 REM lo completaremos mas adelante
1100 RETURN
2000 REM lo completaremos mas adelante
2100 RETURN
3000 REM lo completaremos mas adelante
3100 RETURN
4000 REM lo completaremos mas adelante
4100 RETURN
```

Esto ha sido fácil, ¿verdad? El programa todavía no hace nada práctico, pero sólo es un esqueleto; ahora tendremos que ir colocando los músculos. Los números de línea tan elevados que hemos reservado para las subrutinas nos dejarán espacio amplio para cuando empecemos a programar. Conviene espaciar los subprogramas de 1000 en 1000 para así recordar fácilmente qué líneas hay que listar cuando estemos trabajando con una subrutina concreta. Las subrutinas vacias nos permiten al menos ejecutar el programa.

Observe que ya hemos escrito la rutina de gestión de errores, líneas 5 y 70-150, en previsión de que se produzca algún error cuando la tinta de la pluma sea invisible. No hemos escatimado líneas REM; cuando se está desarrollando un programa largo, al cabo de unos días o semanas es fácil olvidar para qué sirven las diversas subrutinas; las líneas REM nos refrescan la memoria. Ahora bien cuando el programa ya está completo y depurado, las líneas REM lo hacen más largo y más lento. Es conveniente grabar dos versiones del programa, una con líneas REM y otras sin ellas. Si en algún momento se decide introducir alguna modificación en el programa, se trabaja con la versión comentada. La versión más corta es la que se utiliza habitualmente, porque es más rápida.

Bueno, vamos con la subrutina 1000. Se encargará de escribir en la pantalla las

instrucciones de juego y de preguntar al jugador en qué nivel de habilidad quiere jugar. Hagamos la pantalla más atractiva escribiendo con caracteres rojos sobre fondo blanco:

```
1000 PEN 3
1010 INK 0,26
1020 PRINT"En este juego debes intentar bombarde
     ar"
1030 PRINT"todos los edificios para que tu avion
     "
1040 PRINT"pueda aterrizar sin problemas. Pulsa
         b para soltar una bomba."
1050 PRINT:PRINT"En cuanto has soltado una bomba
     , no"
1060 PRINT"puedes tirar la siguiente mientras"
1070 PRINT"la primera no haya llegado al suelo."
1080 PRINT:PRINT"Cual es tu nivel de habilidad?"
1090 INPUT "(de 1=facil a 10=dificil) ",habilida
     d
1100 RETURN
```

Ejecute el programa y compruebe que todo marcha bien. Primero escribe las instrucciones de juego, después capta el nivel de habilidad y termina.

Ahora ya podemos despreocuparnos de esta parte del programa, aunque cabe la posibilidad de que hayamos cometido un error que no descubramos hasta más tarde. Por ejemplo, si hubiéramos puesto 'habillidad' en lugar de 'habilidad', el error se haría patente más tarde, cuando llegase el momento de utilizar esta variable. Es decir, aunque una rutina esté probada, podremos encontrar otros errores cuando combinemos la rutina con otra.

En cuanto a la rutina 2000, hay que tomar muchas decisiones antes de que empiece el juego. Tenemos que elegir el modo de pantalla; tenemos que decidir si este juego es de los que requieren TAG o si, por el contrario, son preferibles las coordenadas de texto. También hemos de diseñar los caracteres que represente el avión y los edificios. Debemos elegir los colores y decidir qué zona de la pantalla vamos a dedicar al cielo y a la hierba. Debemos buscar una forma sencilla de dibujar los edificios y elegir un punto de partida para el avión.

La inicialización de variables consistirá en preparar una variable que indique en cada momento si hay una bomba cayendo, para permitir o impedir el disparo de la siguiente. Hará falta otra variable para indicar si el avión ha chocado con un edificio; en función de ella decidiremos cuándo debe terminar el juego.

Todo esto tiene que estar pensado y decidido antes de que nos pongamos a escribir líneas. Las improvisaciones en el teclado sólo harán que tardemos bastante más en terminar el programa. Hay muchas soluciones posibles para los problemas que hemos planteado; una de ellas es la siguiente:

```
1999 REM preparar el 'cielo'
2000 WINDOW 1,40,1,20
2010 INK 0,2
2020 PAPER 0
2030 CLS
2031 REM preparar la 'hierba'
2040 WINDOW #1,1,40,21,25
2050 INK 2,19
2060 PAPER #1,2
2070 CLS #1
2071 REM definir avion
2080 SYMBOL 240,0,0,192,192,255,0,0,0
2090 SYMBOL 241,0,128,192,224,254,224,192,128
2100 avion$=CHR$(240)+CHR$(241)
2101 REM definir un bloque de edificios
2110 SYMBOL 242,221,221,255,221,221,221,255,221
2120 edificio$=CHR$(242)
2130 bomba$=CHR$(252)
2140 expl$=CHR$(238)
2141 REM numero de edificios en funcion de la ha
     bilidad
2150 numerodeedif=INT(RND*habilidad+5)
2151 REM dibujarlos
2160 FOR numero=1 TO numerodeedif
2161 REM generar aleatoriamente una coordenada x
      de texto
2170 x=INT(RND*20+10)
2171 REM altura en funcion de la habilidad
2180 altura=INT(RND*(habilidad+3)+1)
2181 REM dibujar edificio por bloques
2190 FOR cuenta=0 TO altura
2200 y=20-cuenta:PEN 3
2210 LOCATE x,y
2220 PRINT edificio$;
2230 NEXT
2240 NEXT
2241 REM colocar avion y dibujarlo
2250 avionx=1
2260 aviony=1
2270 INK 1,0
2280 PEN 1
2290 LOCATE avionx,aviony
2300 PRINT avion$;
2301 REM preparar variables para registrar caida
      de bombas, choque, tanteo
2310 choque=0
2320 bomba=0
```

```
2330 tanteo=0
2340 RETURN
```

Ejecute otra vez el programa y compruebe que las dos primeras subrutinas funcionan correctamente. Si los edificios quedaran "colgados" del cielo o la hierba fuera roja, éste sería el momento de corregir tales errores lógicos hasta quedar plenamente satisfechos con el funcionamiento de la subrutina.

La subrutina 3000 es la verdaderamente crucial. Es muy probable que el juego sea suficientemente complicado como para que debamos subdividirlo en secciones más sencillas. De momento, tratemos de identificar el aspecto más importante del juego: cuándo debe terminar. Es evidente que el juego debe continuar mientras el avión no haya aterrizado ni chocado con un edificio. Podemos elegir como final de la pista de aterrizaje, por ejemplo, el borde derecho del suelo:

```
2999 REM el juego continua mientras el avion no
     aterrize o se estrelle
3000 WHILE (avionx<>39 OR aviony<>20) AND choque
     =0
3010 GOSUB 5000
3020 WEND
3021 REM mantener la pantalla final durante 2 se
     gundos
3030 tiempoinicial=TIME
3040 WHILE TIME<tiempoinicial+600
3050 WEND
3060 RETURN
```

La condición de la línea 3000 es algo complicada; sólo podremos comprobarla cuando hayamos desarrollado la subrutina 5000. Aplacemos eso para más tarde y terminemos la última de las subrutinas principales, la de la línea 4000:

```
4000 MODE 1
4010 INK 0,1
4020 INK 1,24
4030 INK 2,2
4040 INK 3,6
4050 IF choque=0 THEN tanteo=tanteo+habilidad
4060 PRINT"Su tanteo para el nivel";habilidad;
4070 PRINT"ha sido";
4080 PEN 3
4090 PRINT tanteo*habilidad
4100 PEN 1
4110 RETURN
```

Esta subrutina restaura los colores normales de pluma y papel y escribe el tanteo. Tal como está el programa, no podemos probar esta subrutina porque la de la línea

3ØØØ no termina nunca. Podemos poner un parche en el programa para "falsificar" el valor de las variables que intervienen en la línea 3ØØØ:

```
5ØØØ choque=1
5100 RETURN
```

Ejecute ahora el programa. El avión no vuela mucho, pero podemos comprobar que la subrutina 4ØØØ funciona. Comprobemos también que el bucle de la 3ØØØ termina cuando el avión está al final de la pista:

```
5ØØØ avionx=39:aviony=2Ø
51ØØ RETURN
```

El programa está casi completo, a falta de la subrutina 5ØØØ, que será la encargada de controlar cada ciclo del juego. Si observamos la pantalla tal como está al principio del juego, vemos que lo primero que hay que hacer es borrar el avión de la posición inicial; después tenemos que:

1) Desplazar el avión y comprobar que su nueva posición no está ocupada por un edificio.
2) Si lo está, visualizar una explosión y hacer choque igual a 1 para que el juego termine.
3) Si hay una bomba cayendo, dibujarla a la altura correcta y visualizar su choque con los edificios o contra el suelo cuando haga falta.
4) Si el jugador pulsa la tecla 'b' y no hay ninguna bomba cayendo, soltar una.

Para que el juego esté más correlacionado con el nivel de habilidad, podemos introducir la siguiente función:

5) Hacer una pausa entre cada dos movimientos cuya duración dependa del nivel de habilidad.

Las acciones 1) y 5) se realizan en todos los casos, así que las programaremos en la subrutina 5ØØØ. Las demás dependerán de las circunstancias; son suficientemente complicadas como para que las releguemos a otras subrutinas. En los capítulo anteriores las sentencias IF ... THEN han llegado a ser muy complejas porque queríamos que las instrucciones que venían después de THEN cupiesen completas en una línea. Si ponemos todas esas instrucciones en una subrutina, las líneas serán más cortas y más fáciles de entender y depurar:

```
4999 REM rutina principal de juego - comienza po
     r borrar el avion
5ØØØ LOCATE avionx,aviony
5Ø1Ø PRINT"  ";
5Ø11 REM mover avion
5Ø2Ø avionx=avionx+1
5Ø21 REM bajar el avion una linea si ha alcanzad
     o el borde derecho
```

```
5030 IF avionx>39 THEN avionx=1:aviony=aviony+1
5031 REM calcular coordenadas graficas de la nue
     va posicion del morro del avion
5040 aviongx=8+avionx*16
5050 aviongy=8+(25-aviony)*16
5060 color=TEST(aviongx,aviongy)
5061 REM fulminar el avion si se detecta un edif
     icio
5070 IF color<>0 THEN GOSUB 6000:RETURN
5071 REM mover la bomba si esta cayendo
5080 IF bomba=1 THEN GOSUB 7000
5081 REM examinar teclado
5090 respuesta$=INKEY$
5091 REM no dejar caer la bomba si hay otra caye
     ndo
5100 IF respuesta$="b" AND bomba=0 THEN GOSUB 80
     00
5101 REM dibujar avion en la nueva posicion
5110 PEN 1
5120 LOCATE avionx,aviony
5130 PRINT avion$;
5131 REM hacer pausa de 0 a 3 centesimas de segu
     ndo
5140 tiempoinicial=TIME
5150 WHILE TIME<tiempoinicial+10-habilidad
5160 WEND
5170 RETURN
5999 REM volar avion
6000 REM lo completaremos mas adelante
6100 RETURN
7000 REM lo completaremos mas adelante
7100 RETURN
8000 REM lo completaremos mas adelante
8100 RETURN
```

Observe la línea 5070: ésta es la primera vez que ponemos más de un RETURN en una subrutina. El significado de la línea 5070 es: "si el avión ha chocado, ir a la subrutina 6000; al retornar, terminar también esta subrutina". Después de todo, si el avión ha chocado el juego tienen que terminar; no debemos perder el tiempo realizando las acciones 3) a 5).

La fórmula de la línea 5040 no es la que antes utilizábamos para calcular la coordenada gráfica X. El avión está formado por dos caracteres; la coordenada avionx es la coordenada de texto del carácter de la izquierda. Si en TEST ponemos las coordenadas del centro de la siguiente posición de texto, lo que estamos comprobando es el color del morro del avión. La fórmula de la línea 5040 tiene esto en cuenta y realiza el cálculo correcto.

Ejecute el programa. Ahora podemos probar y depurar esta rutina de movimiento del avión. Este se desplaza de izquierda a derecha correctamente, pero cuando choca con un avión, línea 5070, continúa como si tal cosa; esto se debe a que todavía no hemos desarrollado la subrutina 6000, cuya misión sera hacer 'choque' igual a 1:

```
6000 choque=1
```

Ejecute ahora el programa. Cuando el avión choca con un edificio, el juego termina y el ordenador nos informa del bajo tanteo obtenido.

También debemos comprobar que el programa funciona bien cuando conseguimos aterrizar con el avión. La forma más sencilla de hacerlo es no dibujar ningún edificio:

```
2143 GOTO 2250
```

Esta instrucción provoca el salto del programa a la línea especificada, impidiendo así el dibujo de los edificios. Muchos programas utilizan indiscriminadamente la instrucción GOTO ("ir a"), y esto los hace difíciles de seguir y depurar. GOTO tiene sus aplicaciones, como en este caso, pero es una instrucción que se debe prodigar lo menos posible. Nosotros preferimos utilizarla solamente como ayuda en la depuración de programas; en este caso la borraremos en cuanto comprobemos el funcionamiento de la subrutina 5000.

Hemos empezado a desarrollar la subrutina 6000, así que vamos a completarla:

```
5999 REM fulminar el avion
6000 LOCATE avionx,aviony
6010 PRINT expl$;
6011 REM destellos de colores
6020 INK 3,6,26
6021 REM indicador de si el avion ha chocado
6030 choque=1
6040 RETURN
```

Ejecute el programa para comprobar que esta sección funciona.

Vamos a ocuparnos de la subrutina 8000, que deja caer la bomba:

```
7999 REM dejar caer la bomba cuando se ha pulsad
     o 'b'
8000 bombax=avionx
8001 REM la bomba debe estar una linea mas abajo
      que el avion
8010 bombay=aviony+1
8011 REM no dejarla caer si el avion esta en la
     hierba
8020 IF bombay>20 THEN RETURN
8021 REM indicador de que la bomba esta cayendo
```

```
8030 bomba=1
8031 REM dibujar la bomba en su posicion de part
     ida
8040 LOCATE bombax,bombay
8050 PEN 1
8060 PRINT bomba$;
8070 RETURN
```

Ejecute el programa y compruebe que efectivamente se puede disparar la bomba. Quedará suspendida en el aire y será imposible disparar otras porque la variable 'bomba' está a 1. Esto demuestra que las condiciones de la línea 5100 son las necesarias para impedir el disparo de una bomba mientras hay otra cayendo y que la subrutina 8000 funciona correctamente.

La subrutina 7000 controla la caída de la bomba. Será similar a la 5000, que controla el movimiento del avión:

1) Bajar la bomba y comprobar si en la nueva posición hay un edificio.
2) Si la bomba a alcanzado un edificio, visualizar una explosión, poner a 0 la variable 'bomba' y terminar la subrutina.
3) En caso contrario, dibujar la bomba en la nueva posición.

Análogamente a lo que ocurría en 5000, las acciones 1) y 3) serán las más frecuentes y por eso las programaremos en esta subrutina. En cambio, 2) deberá ser relegada a otra subrutina:

```
6999 REM rutina de caida de bomba - comenzar por
      borrar la bomba
7000 LOCATE bombax,bombay
7010 PRINT" ";
7011 REM bajar la bomba una linea
7020 bombay=bombay+1
7021 REM calcular las coordenadas graficas de la
      nueva posicion de la bomba
7030 bombagx=8+(bombax-1)*16
7040 bombagy=8+(25-bombay)*16
7050 color=TEST(bombagx,bombagy)
7051 REM que estalle la bomba si se ha detectado
      un edificio
7060 IF color<>0 THEN GOSUB 9000:RETURN
7061 REM dibujar la bomba en la nueva posicion
7070 PEN 1
7080 LOCATE bombax,bombay
7090 PRINT bomba$;
7100 RETURN
9000 REM lo completaremos mas adelante
9100 RETURN
```

Comparando esta subrutina con la 5ØØØ se puede comprobar que son muy parecidas. Ejecute el programa. La bomba ya no se queda paralizada, sino que continúa cayendo hasta que choca con un edificio o contra el suelo. No podemos disparar más bombas porque no hemos escrito la subrutina 9ØØØ. Una de sus misiones será poner a cero la variable 'bomba' para permitir el disparo de la siguiente:

```
9000 bomba=0
```

La subrutina 7ØØØ parece funcionar bien, aunque la bomba se queda en la pantalla porque borrarla es misión de la subrutina 9ØØØ. Otra cosa que tiene que hacer la subrutina 9ØØØ es visualizar una explosión en el punto de choque de la bomba. Si el choque se ha producido contra un edificio, tenemos que actualizar el tanteo:

```
9000 REM escribiremos esta linea enseguida
9001 REM dibujamos la explosion y hacemos que de
     stelle rapidamente
9010 LOCATE bombax,bombay
9020 INK 1,0,6
9030 PEN 1
9040 PRINT expl$;
9041 REM devolver pluma a lo normal
9050 INK 1,0
9051 REM la bomba ya no cae - poner a cero el in
     dicador
9060 bomba=0
9061 REM incrementar tanteo si se la bomba ha al
     canzado un edificio
9070 IF color=3 THEN tanteo=tanteo+1
9071 REM borrar la explosion
9080 LOCATE bombax,bombay
9090 PRINT" ";
9100 RETURN
```

Al ejecutar el programa descubrimos un error lógico: cada vez que la bomba llega al suelo, la pantalla se desplaza hacia arriba. Esto ocurre porque dibujamos la explosión cuando 'bombay' es 21, y eso está fuera de la ventana. El problema se resuelve fácilmente de la siguiente forma:

```
8999 REM volver a meter la bomba en la ventana s
     i se sale
9000 IF bombay>20 THEN bombay=20
```

Ejecutando el programa unas cuantas veces podríamos convencernos de que no quedan errores importantes. Pero no seamos tan optimistas: todavía no hemos probado el programa en todas las condiciones posibles. De hecho, quedan errores.

Uno que el lector puede haber descubierto es el siguiente: cuando el avión está

inmediatamente por encima de un edificio en el momento de disparar la bomba, ésta destruye varios bloques del edificio, no uno sólo. El problema está en la rutina 8000, en la que no hemos previsto esta posibilidad. Tenemos que comprobar el color de la nueva posición de la bomba en todos los casos, aun cuando acabemos de dispararla. Para ello podríamos repetir las líneas 7030 a 7060 en la subrutina 8000, pero hay otra forma mejor de hacerlo.

Una de las ventajas de las subrutinas es que nos permiten subdividir el programa en trozos manejables. Otra ventaja muy importante es que una misma rutina puede ser utilizada por diversas secciones del programa. En lugar de tener un grupo de líneas repetido varias veces a lo largo del programa, podemos poner esas líneas en una subrutina e invocar la subrutina cada vez que la necesitemos. Esto es justamente lo que vamos a hacer ahora. En la figura 34 vemos que hay dos subrutinas con necesidades comunes a ambas.

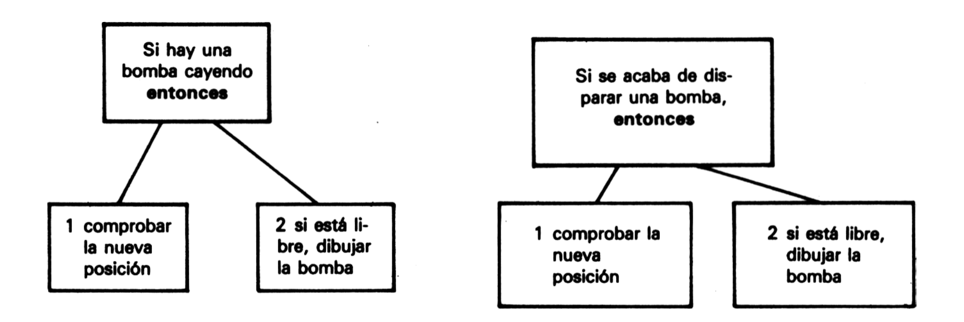


Figura 34. Subrutinas.

Vamos a quitar las líneas que calculan las coordenadas gráficas y a ponerlas en una subrutina que empiece en la línea 10000. Esta subrutina tendrá que ser invocada desde las subrutinas 7000 y 8000:

```
7020 bombay=bombay+1
7021 REM calcular coordenadas graficas de la nue
     va posicion de la bomba
7030 GOSUB 10000
7051 REM que estalle la bomba si se ha detectado
      un edificio
7060 IF color<>0 THEN GOSUB 9000:RETURN
8030 bomba=1
8031 REM dibujar la bomba en su posicion de part
     ida
8035 GOSUB 10000
8036 IF color<>0 THEN GOSUB 9000:RETURN
8040 LOCATE bombax,bombay
9100 RETURN
10000 bombagx=8+(bombax-1)*16
```

```
10010 bombagy=8+(25-bombay)*16
10020 color=TEST(bombagx,bombagy)
10030 RETURN
```

La forma más rápida de comprobar que esto a corregido el defecto es dibujar todos los edificios con la altura máxima:

```
2182 altura=18
```

Ejecute el programa y dispare varias bombas. Sólo se destruye el bloque superior del edificio, luego el problema está resuelto.

Ahora deberíamos comprobar todo el resto del programa, porque frecuentemente al corregir un error se introducen otros nuevos.

Aunque la subrutina 10000 ha corregido el defecto, las subrutinas 7000 y 8000 tienen más tareas comunes que el cálculo de las coordenadas gráficas. Intente poner en la subrutina 10000 *todas* las líneas que son similares en las subrutinas 7000 y 8000.

Resumen

Lamentablemente, en un libro dedicado a describir y explicar las instrucciones de BASIC no es posible dedicar al tema de la planificación de programas el espacio que merece. La principal conclusión que debemos deducir de este ejemplo es que la planificación permite identificar y preparar las subrutinas en que se debe dividir el programa. Las subrutinas se escriben entonces una a una, lo que da la oportunidad de probarlas y depurarlas antes de pasar a la siguiente. Los errores restantes serán los que surjan al ensamblar unas subrutinas con otras y resultarán más fácil de detectar.

Los diagramas también son de gran ayuda en la planificación de programas. Hay métodos sistemáticos para la elaboración de diagramas; en general basta con escribir la descripción de los diversos bloques de acciones en unos rectángulos e interconectar los rectángulos con líneas que indiquen qué subrutinas llaman a qué otras. Es conveniente analizar el plan del programa para determinar qué acciones se pueden programar con una sucesión de líneas, cúales se deberán realizar con bucles y dónde habrá que tomar decisiones; de este análisis se suele deducir qué funciones son susceptibles de ser encomendadas a subrutinas.

No se deje desanimar por los inevitables errores. En el programa que hemos desarrollado no ha funcionado completo al primer intento, pero al dividirlo en subprogramas nos nos ha sido difícil depurarlo.

Ejercicios

1. Modifique el programa anterior para que visualice y mantenga actualizado un tanteador a lo largo del juego.

2. Limite el número de bombas disponibles, en función del nivel de habilidad. Mantenga informado al jugador del número de bombas restantes. (El número y altura de los edificios imponen un mínimo al número de bombas; con menos bombas el avión no podrá llegar a aterrizar.)
3. Permita que el jugador pueda disparar una megabomba cuyo efecto sea destruir completamente el edificio sobre el que caiga. (No olvide que tendrá que ajustar el tanteo en función de la altura del edificio.)
4. Reescriba el programa de juego para dos jugadores del capítulo 8 de forma que quede estructurado en subrutinas.

10

Música y sonidos

Un aspecto que hemos omitido completamente en nuestros programas anteriores es el sonido. Las funciones de sonido del Amstrad son muy completas. Si le interesa la generación de música con el ordenador, en los apéndices del Manual del Usuario encontrará información muy completa sobre las instrucciones de sonido. Producir notas aisladas de una frecuencia determinada es muy sencillo. El siguiente programa facilita la experimentación con la instrucción SOUND:

```
5 ON BREAK GOSUB 500
10 MODE 1
20 GOSUB 1000
30 GOSUB 2000
200 GOSUB 500
300 END
500 MODE 1
505 INK 1,24
510 PEN 1
520 PAPER 0
530 END
1000 PAPER 1
1010 CLS
1020 PEN 3
1030 PRINT"Este programa le da una oportunidad d
     e"
1040 PRINT"experimentar con la orden SOUND."
1050 PRINT
1060 PRINT"Pulse cualquiera de las teclas 1 a 7"
1070 PRINT"para poner a cero el canal."
1080 PRINT
1090 PRINT"Pulse 's' para subir el tono de la no
     ta"
1100 PRINT"y 'l' para bajarlo."
1110 PRINT
1120 PRINT"Pulse 'f' para terminar."
1130 PRINT
1140 PRINT"La orden SOUND estara siempre en la"
```

```
1150 PRINT"pantalla para que usted la observe al
     "
1160 PRINT"mismo tiempo que la escucha."
1170 PRINT
1180 INPUT"Pulse ENTER cuando este preparado.",e
     nter
1190 RETURN
2000 PEN 1
2010 PAPER 0
2020 CLS
2030 CLS
2040 respuesta$=""
2050 tono=478
2060 canal=1
2070 x=4
2080 canalx=10
2090 tonox=12
2100 y=12
2110 INK 1,0
2120 PEN 3
2130 LOCATE x,y
2140 PRINT"SOUND";
2150 WHILE respuesta$<>"f"
2160 respuesta$=""
2170 WHILE respuesta$=""
2180 respuesta$=INKEY$
2190 WEND
2200 codigo=ASC(respuesta$)
2210 IF codigo>48 AND codigo<56 THEN canal=codig
     o-48
2220 IF codigo=98 AND tono<4000 THEN tono=tono+1
2230 IF codigo=115 AND tono>100 THEN tono=tono-1
2240 PEN 1
2250 LOCATE canalx,y
2260 PRINT canal;
2270 PEN 3
2280 LOCATE tonox,y
2290 PRINT tono;
2300 SOUND canal,tono
2310 WEND
2320 RETURN
```

El sonido es generado por la línea 2300. Los dos parámetros que siguen a la palabra clave SOUND son obligatorios. El primero es el *número de selección de canales*. El ordenador tiene tres canales de sonido: A, B y C. Cada combinación de canales está representada por un número del margen 1-255. En la figura 35 se puede reco-

nocer la analogía entre estos números y los de la instrucción SYMBOL. Para seleccionar el canal A utilizaríamos el número 1; para el C, el 4. Pero para seleccionar simultáneamente los canales A y C necesitamos la suma de 1 y 4, es decir, el 5. Para seleccionar los tres, el número será 1+2+4, o sea, el 7.

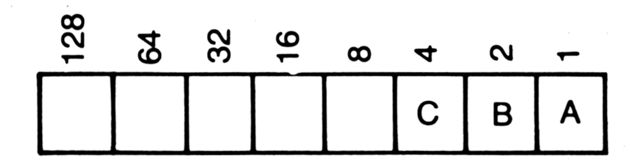


Figura 35. Números de selección de canales.

A estos números se puede sumar otros mayores para controlar la sincronización entre canales. El control completo de los canales no es difícil, pero queda fuera del alcance de este libro. En cualquier caso, está bastante bien explicado en el Manual del Usuario.

El segundo parámetro de SOUND establece el *tono* de la nota. El tono está representado por un número del margen Ø-4Ø95. Cuanto menor es el número, más aguda es la nota. En el apéndice VII del Manual del Usuario se da una lista completa de los números necesarios para generar las notas de varias octavas. Consulte esa lista si pretende programar algo más que unas notas sueltas. En la línea 2Ø5Ø del programa se establece el número de tono en 478, que corresponde a la nota DO media.

Usted puede variar la selección de los canales y el tono en el transcurso del programa. El número de tono se cambia de unidad en unidad en las líneas 222Ø y 223Ø. La amplitud de la variación se puede modificar fácilmente editando estas líneas. Los números de tono están limitados al margen de 1ØØ a 4ØØØ para asegurar que no se salen del margen permisible (Ø a 4Ø95), pero usted puede cambiar estos límites. La duración de las notas es de aproximadamente Ø.2 segundos.

El programa examina continuamente el teclado, línea 22ØØ, y modifica la selección de canales o el tono si se pulsan las teclas adecuadas, líneas 221Ø a 223Ø. Se utilizan los códigos ASCII, pero también podríamos haber puesto la línea 221Ø de la siguiente forma:

```
2210 IF INSTR(respuesta$,"1234567")>0 THEN numer
     o=VAL(respuesta$):canal=numero
```

El programa muestra continuamente el número de selección de canales y el número de tono para que el usuario pueda correlacionar fácilmente estos valores con los efectos audibles.

Ejercicios

1. Escriba un programa que lea en las líneas DATA los datos necesarios para interpretar una melodía sencilla utilizando repetidamente la instrucción SOUND.
2. Incluya algunos sonidos en el juego para jugadores del capítulo 8. Haga sonar una nota por cada movimiento, con diferentes números de tono para cada jugador. También utilizar un tono distinto para cada dirección de movimiento.
3. Modifique el programa del ejercicio anterior para que interprete una melodía cuando un jugador choque con el otro o con una pared.

Parámetros opcionales

La instrucción SOUND puede llevar hasta 5 parámetros opcionales. Indican, respectivamente, lo siguiente: la duración de la nota; el volumen; si el volumen de la nota debe ser controlado por una envolvente de volumen; si el tono debe ser controlado por una envolvente de tono; la cantidad de ruido que se debe mezclar con la nota. Estudiaremos más adelante las envolventes de volumen y de tono. Por ahora pondremos a cero estos parámetros para observar el efecto de los restantes. Añada las siguientes líneas al programa anterior:

```
1111 PRINT"Pulse 'i' para hacer la nota mas larg
     a"
1112 PRINT"y 'd' para hacerla mas corta."
1113 PRINT
1114 PRINT"Pulse 'q' para bajar el volumen"
1115 PRINT"y 'r' para subirlo"
1116 PRINT
1117 PRINT"Pulse 'n' para subir el nivel de ruid
     o."
1118 PRINT"y la barra espaciadora para bajarlo."
1119 PRINT
2061 duracion=20
2062 volumen=12
2063 ruido=0
2070 x=4
2080 canalx=10
2090 tonox=12
2091 duracionx=16
2092 volumenx=20
2093 ruidox=27
2141 LOCATE 23,y
2142 PRINT"0 0";
```

```
2360 LOCATE volumenx,y
2370 PRINT volumen;
2380 PEN 1
2390 LOCATE ruidox,y
2400 PRINT ruido;
2410 SOUND canal,tono,duracion,volumen,0,0,ruido
2420 WEND
2430 RETURN
2231 IF codigo=105 THEN duracion=duracion+1
2232 IF codigo=100 THEN duracion=duracion-1
2233 IF codigo=113 AND volumen>0 THEN volumen=vo
     lumen-1
2234 IF codigo=114 AND volumen<15 THEN volumen=v
     olumen+1
2235 IF codigo=110 AND ruido<31 THEN ruido=ruido
     +1
2236 IF codigo=32 AND ruido>0 THEN ruido=ruido-1
2310 REM para borrar la antigua 2310
2320 PEN 1
2330 LOCATE duracionx,y
2340 PRINT duracion;
2350 PEN 3
```

Ahora el programa permite modifcar todos los parámetros y los exhibe en la pantalla.

La duración de las notas se mide en centésimas de segundo; la línea 2Ø61 establece una duración de Ø.2 segundos, que es la que el ordenador supondría si no incluyésemos este parámetro. El volumen puede variar entre Ø (silencio) y 15; en la línea 2Ø62 establecemos el volumen estándar, 12. El ruido puede variar entre Ø y 31; el valor por defecto es Ø. Añadir ruido a una nota puede parecer contraproducente, pero es la base de los efectos sonoros especiales.

Ejercicios

1. Introduzca en el programa del bombardero algunos efectos sonoros. Programe una nota de duración adecuada, mezclada con ruido, para simular las explosiones. Haga sonar una nota con cada movimiento del avión, pero de forma que la duración esté en función del nivel de habilidad para que el programa no resulte indebidamente lento. Aumente el tono de la nota cada vez que el avión baja una línea.

La envolvente de volumen

La envolvente de volumen controla la forma en que el volumen varía durante el tiempo que está sonando la nota. Se pueden definir hasta 15 envolventes distintas. Una vez definida una envolvente, el ordenador almacena sus características hasta que se lo desconecta. Para utilizar una envolvente, se cita su número como quinto parámetro en la instrucción SOUND. Por ejemplo, SOUND 1,478,2Ø,12 hace sonar la nota DO media por el canal A durante Ø.2 segundos a volumen 12. Al incluir como quinto parámetro el número 3, SOUND 1,478,2Ø,12,3 hace sonar la misma nota, pero ahora el 12 especifica sólo el volumen inicial; la evolución del volumen con el tiempo es controlada por la envolvente de volumen número 3, que habrá sido definida con anterioridad.

Añada al programa las siguientes líneas, con las que podrá definir envolventes de volumen y observar sus efectos:

```
15 GOSUB 6ØØ
6ØØ PRINT"Esta subrutina le permite definir una"
61Ø PRINT"envolvente de volumen antes de probar"
62Ø PRINT"diferentes notas con ella."
63Ø PRINT
64Ø INPUT"Numero de envolvente (1-15)";envolvent
    evol
65Ø INPUT"Numero de escalones (Ø-27)";numescvol
66Ø INPUT"Altura de escalon (-128 - +127)";altes
    cvol
67Ø INPUT"Tiempo de pausa (Ø-255)";pausavol
68Ø ENV envolventevol,numescvol,altescvol,pausav
    ol
69Ø PRINT
7ØØ INPUT "Pulse ENTER para continuar",enter
71Ø RETURN
2121 LOCATE x,y-2
2122 PRINT"ENV";envolventevol;",";numescvol;",";
     altescvol;",";pausavol
241Ø SOUND canal,tono,duracion,volumen,envolvent
     evol,Ø,ruido
```

La forma de la envolvente de volumen es la siguiente:

ENV número de la envolvente,número de escalones,altura de cada escalón, tiempo de pausa.

El número de envolvente es aquél por el cual se la invoca en la instrucción SOUND.

El número de escalones especifica en cuántas etapas queremos que evolucione la nota.

La altura de cada escalón especifica la variación del volumen con respecto al escalón anterior.

El tiempo de pausa especifica la duración de cada escalón en unidades de centésimas de segundo.

Por ejemplo, ENV 1,1ØØ,−5,1 describe una envolvente compuesta por 1ØØ etapas de una centésima de segundo cada una, con un descenso de volumen de una a otra de 5 unidades. A pesar de que el volumen de la nota está controlado por la envolvente, para apreciar la envolvente completa la duración especificada en SOUND tiene que ser suficiente. Así, con SOUND 1,478,2Ø,12,1 perderíamos parte del efecto ENV 1,1ØØ,−5,1 ya que la envolvente dura 1 segundo (1ØØ centésimas) y en SOUND hemos especificado una duración de sólo Ø.2 segundos (2Ø centésimas).

La mejor forma de comprender los efectos de las envolventes de volumen es experimentar con el programa anterior. Defina una envolvente de volumen y ejecute una orden SOUND adecuada; así podrá apreciar cómo afecta la envolvente a la nota resultante. Vaya cambiando después los parámetros uno a uno, por ejemplo, el tono o el nivel de ruido, y observe los diferentes efectos. Cuando consiga algún efecto interesante, tome nota de todos los parámetros de ENV y SOUND con los que lo ha producido. En poco tiempo elaborará un amplio catálogo de efectos sonoros que más tarde puede incorporar a otros programas. En el capítulo siguiente explicaremos cómo se pueden grabar estos datos en cinta.

La envolvente de tono

La estructura es semejante a la de la envolvente de volumen:

ENT número de la envolvente,número de escalones,altura de cada escalón, tiempo de pausa.

El número de envolvente puede variar también entre 1 y 15, pero ahora se pueden poner números negativos para hacer que la envolvente se repita hasta el final de la duración especificada en SOUND.

El número de escalones especifica en cuántas etapas queremos que evolucione la nota.

La altura de cada escalón especifica la variación del tono con respecto al escalón anterior.

El tiempo de pausa especifica la duración de cada escalón en unidades de centésimas de segundo.

Añada al programa las siguientes líneas, con las que podrá definir envolventes de tono y observar sus efectos:

```
16 GOSUB 8ØØ
8ØØ PRINT"Esta subrutina le permite definir una"
81Ø PRINT"envolvente de tono."
82Ø PRINT
83Ø PRINT"Numero de envolvente (1-15, numeros ne
   -"
835 INPUT"gativos para que sea de repeticion)";e
   nvolventetono
```

```
840 INPUT"Numero de escalones (0-239)";numescton
    o
850 INPUT"Altura de escalon (-128 - +127)";altes
    ctono
860 INPUT"Tiempo de pausa (0-255)";pausatono
870 ENT envolventetono,numesctono,altesctono,pau
    satono
880 PRINT
890 INPUT "Pulse ENTER para continuar",enter
900 RETURN
2123 LOCATE x,y-4
2124 PRINT"ENT";envolventetono;",";numesctono;",
     ";altesctono;",";pausatono
2410 SOUND canal,tono,duracion,volumen,envolvent
     evol,ABS(envolventetono),ruido
```

Al experimentar con envolventes de tono conviene no utilizar las de volumen para no confundir sus efectos.

11

Ficheros

En este capítulo vamos a explicar cómo se puede utilizar el ordenador para grabar datos en un fichero y luego recuperarlos. Un fichero de ordenador es muy similar al tradicional fichero de fichas de cartulina: consiste en una serie de fichas o *registros* que contienen información homogénea. Los registros se dividen en *campos*. Por ejemplo, en un fichero de clientes un registro es el conjunto de datos relativos a un cliente determinado; los campos de que consta el registro son; nombre, señas, número de teléfono, saldo de su cuenta, etc.

Lo habitual es disponer de un programa que crea el fichero y lo graba en la cinta. Otro programa puede encargarse de recuperar el fichero, ampliarlo o modificarlo y luego volver a grabarlo. Las operaciones típicas que se realizan con los ficheros son, pues, las siguientes:

1) Grabar un fichero en cinta.
2) Cargar un fichero en la memoria del ordenador leyéndolo de la cinta.
3) Añadir nuevos registros al fichero.
4) Borrar registros.
5) Modificar registros.

En un sistema de archivo en cinta, como es el del Amstrad, es teóricamente posible manejar los registros uno por uno, pero ello requiere tal manipulación de cintas, que en la práctica el método es inviable.

Lo que vamos a hacer es grabar y leer ficheros completos, en vez de registros aislados. Una vez cargado un fichero en la memoria, podremos consultarlo, ampliarlo o modificarlo, y luego lo volveremos a grabar completo en la cinta. Bastará con dos cassettes como máximo: una para leer la versión antigua del fichero; la otra para recibir la versión actualizada.

Todas estas ideas quedarán más claras cuando las ilustremos con un ejemplo. Vamos a desarrollar un sistema de archivo de nombres y números de teléfono.

Creación del fichero

El fichero se puede crear con el siguiente programa:

```
10 MODE 1
20 GOSUB 1000
30 GOSUB 2000
```

```
60 END
1000 PRINT"Puede usar este programa para crear u
     n"
1010 PRINT"fichero de nombres y numeros de tele-
     "
1020 PRINT"fono. La tecnica es de interes genera
     l,"
1030 PRINT"por lo que usted puede adaptar el pro
     -"
1040 PRINT"grama a cualquier tipo de datos."
1050 PRINT
1070 PEN 2
1080 PRINT"Dentro de un momento le pedire los no
     m-"
1090 PRINT"bres y numeros de telefono de sus"
1100 PRINT"amigos. Escribalos y, cuando quiera"
1110 PRINT"acabar, introduzca xxx,xxx ."
1120 PRINT
1130 PRINT"Los datos seran grabados en cinta."
1140 PRINT"Ponga en el magnetofono una cinta"
1150 PRINT"virgen. Yo le avisare cuando deba"
1160 PRINT"pulsar los botones de RECORD y PLAY."
1170 PRINT
1180 INPUT"Pulse ENTER cuando este preparado.",e
     nter
1190 RETURN
2000 CLS
2010 INPUT"Que nombre quiere dar a este fichero
        ";fichero$
2020 OPENOUT fichero$
2030 PRINT"Escriba un nombre y un numero de "
2040 PRINT"telefono."
2050 PRINT
2060 PEN 3
2070 INPUT"Nombre, numero de telefono:
        ",nombre$,telefono$
2080 WHILE nombre$<>"xxx"
2090 WRITE #9,nombre$,telefono$
2100 PRINT
2110 INPUT"Nombre, numero de telefono:
        ",nombre$,telefono$
2120 WEND
2130 CLOSEOUT
2140 PRINT
2150 PEN 1
2160 PRINT"Sus datos han sido grabados en cinta.
     "
```

```
2170 PRINT"Ahora puede leer los datos del ficher
     o"
2180 PRINT"con el siguiente programa."
2190 RETURN
```

La subrutina de la línea 1000 explica al usuario el manejo del programa. La de la línea 2000 crea el fichero y graba los datos en él. El procedimiento es muy sencillo. En la línea 2010 se capta por el teclado el nombre del fichero. La línea 2020 lo abre con OPENOUT (“abrir en dirección de salida”) y lo deja preparado para recibir los datos.

Los datos son captados y enviados a la cinta por el bucle de las líneas 2080 a 2120. La línea 2090 contiene la instrucción WRITE #9 (*write* significa “escribir”), con la que se escriben los datos en la cinta. En realidad, el ordenador no graba los datos en la cinta inmediatamente después de captarlos, sino que los guarda temporalmente en una zona de la memoria reservada al efecto y denominada *tampón*. Cuando se llena el tampón es cuando se realiza el proceso físico de la grabación en cinta.

La línea 2130 es muy importante. Informa al ordenador de que hemos terminado de preparar datos y queremos que cierre el fichero. Como consecuencia de esta instrucción, el ordenador vacía los datos que puediera haber en el tampón y los graba en la cinta.

Una vez creado el fichero, podemos apagar el ordenador, sabiendo que aunque se borre la RAM, los datos están a salvo en la cinta.

Lectura del fichero

Vamos a ampliar el programa anterior para que lea el fichero inmediatamente después de crearlo. No es esto lo que haríamos en la práctica, pero nos demostrará que el fichero ha quedado grabado:

```
40 GOSUB 3000
50 GOSUB 4000
3000 PEN 2
3010 PRINT
3020 PRINT"Este programa sirve para cargar en"
3030 PRINT"memoria un fichero grabado en cinta."
3040 PRINT
3050 PEN 2
3060 PRINT"Ponga en el magnetofono la cinta en l
     a"
3070 PRINT"que este el fichero."
3080 PRINT
3090 INPUT"Pulse ENTER cuando este preparado.",e
     nter
3100 RETURN
```

```
4000 CLS
4010 INPUT"Como se llama el fichero
     ";nombrefichero$
4020 OPENIN nombrefichero$
4030 PRINT"Esta es la lista de todos los nombres
     y"
4040 PRINT"numeros de telefono:"
4050 WHILE NOT EOF
4060 INPUT #9,amigo$,telefono$
4070 PRINT
4080 PRINT amigo$,telefono$
4090 WEND
4100 CLOSEIN
4110 PRINT
4120 PRINT"Eso es todo!"
```

Para que el ordenador pueda leer el fichero, tiene que saber cómo se llama (línea 4010). Como vamos a leer del fichero, tendremos que abrirlo en dirección de entrada (OPENIN, línea 4020).

No es probable que recordemos cuántos datos hemos grabado en el fichero. Entonces ¿cómo sabe el Amstrad cuántos datos tiene que leer?

Muy sencillo: le diremos que lea datos hasta que llegue al final del fichero. El ordenador graba automáticamente al final de cada fichero una ''señal de fin de fichero''(EOF). No nos interesa demasiado cómo lo hace; nos basta con saber que podemos aprovechar esa señal. Las líneas 4050 a 4090 equivalen a lo siguiente:

WHILE (mientras) no se haya alcanzado el final del fichero (EOF)
leer un registro
y escribirlo
WEND

INPUT #9 es la instrucción con la que se lee de la cinta.

Ya hemos leído el fichero, pero nos encontramos con que hemos perdido todos los registros menos el último. Cada vez que la línea 4060 lee un registro , el ordenador ''olvida'' los valores anteriores de amigo$ y telefono$. ¿Cómo podemos leer el fichero y almacenar al mismo tiempo los datos en la memoria para después examinarlos con tranquilidad?

Tenemos que asignar los datos a elementos de un par de listas según los vayamos leyendo:

```
4000 CLS
4010 INPUT"Como se llama el fichero
     ";nombrefichero$
4015 DIM amigo$(100),telefono$(100)
4016 cuenta=1
4020 OPENIN nombrefichero$
4030 PRINT"Esta es la lista de todos los nombres
     y"
```

```
4040 PRINT"numeros de telefono:"
4050 WHILE NOT EOF
4060 INPUT #9,amigo$(cuenta),telefono$(cuenta)
4070 PRINT
4080 PRINT amigo$(cuenta),telefono$(cuenta)
4085 cuenta=cuenta+1
4090 WEND
4100 CLOSEIN
4110 PRINT
4120 PRINT"Eso es todo!"
```

La línea 4015 dimensiona las listas para que puedan contener hasta 100 nombres y números de teléfono. Podríamos haberlas hecho más grandes, pues el único límite es el impuesto por la capacidad de memoria.

El programa utiliza un contador para llevar la cuenta del número de registros leídos y al mismo tiempo para hacer de subíndice en amigo$( ) y telefono$( ). Cuando termina la lectura del fichero, toda la información ha quedado almacenada en estas dos listas.

Ya hemos visto en un capítulo anterior cómo se puede buscar un elemento en una lista. En el programa del restaurante de comida rápida, el usuario escribía el nombre del artículo y el programa lo buscaba en la lista. El precio se encontraba en el elemento de igual subíndice de la lista de precios.

Podríamos utilizar la misma técnica para buscar el número de teléfono después de captar por el teclado un nombre. Pero en vez de seguir ampliando el programa vamos a reorganizarlo.

Programa controlado por menú

El siguiente programa utiliza el fichero de nombres y números de teléfono y demuestra cómo se puede manipular el fichero y volver a grabarlo después de modificado. Este programa puede ser adaptado para que controle cualquier otro tipo de ficheros.

El programa tiene que pedir al usuario que elija una de las varias opciones que puede ofrecerle; en casos como éste lo más conveniente es ofrecer las opciones en forma de *menú*:

```
10 MODE 1
20 GOSUB 1000
30 END
1000 respuesta$=""
1010 cargafichero=0
1020 WHILE respuesta$<>"5"
1030 PEN 3
1040 PRINT
1050 PRINT"Elija 1 - 5:"
1060 PEN 1
```

```
1070 PRINT:PRINT"1. Cargar fichero"
1080 PRINT:PRINT"2. Grabar fichero"
1090 PRINT:PRINT"3. Buscar en el fichero"
1100 PRINT:PRINT"4. Corregir fichero"
1110 PRINT:PRINT"5. Fin"
1120 PRINT
1130 PEN 3
1140 PRINT"Cual?"
1150 WHILE respuesta$="":respuesta$=INKEY$:WEND
1160 IF respuesta$="1" THEN GOSUB 2000
1170 IF respuesta$="2" THEN GOSUB 3000
1180 IF respuesta$="3" THEN GOSUB 4000
1190 IF respuesta$="4" THEN GOSUB 5000
1200 IF respuesta$<>"" THEN CLS
1210 IF respuesta$<>"5" THEN respuesta$=""
1220 WEND
1230 RETURN
```

En un programa más complejo, el menú principal conduciría a una serie de submenúes, en los cuales se ofrecerían al usuario nuevas opciones. Se suele procurar que el usuario no tenga que escribir mucho para realizar la elección; lo idóneo es que baste con que pulse una tecla. Las opciones suelen estar numeradas, como en este caso.

La subrutina 2000 carga el fichero:

```
2000 CLS
2010 INPUT"Cual es el nombre del fichero
     ";fichero$
2020 cuenta=1
2030 IF cargafichero=1 THEN ERASE nombre$,telefo
     no$
2040 DIM nombre$(100),telefono$(100)
2050 OPENIN fichero$
2060 WHILE NOT EOF
2070 INPUT #9,nombre$(cuenta),telefono$(cuenta)
2080 cuenta=cuenta+1
2090 WEND
2100 CLOSEIN
2110 PRINT
2120 PEN 2
2130 PRINT"El fichero ha quedado cargado."
2140 INPUT"Pulse ENTER para volver al menu.",ent
     er
2150 cargafichero=1
2160 RETURN
```

Estas instrucciones son similares a las correspondientes en el programa de la sección anterior. Hemos incluido la línea 2030 por si queremos volver a utilizar la subrutina para leer otro fichero. Cuando se ha ejecutado una vez la rutina, el ordenador ya "conoce" las listas nombre$( ) y telefono$( ), y entonces la línea 2040 provoca un error. La línea 2150 pone a 1 la variable 'cargafichero' para que cuando se vuelva a ejecutar la línea 2030 la orden ERASE borre las listas y la línea 2040 no dé problemas.

La subrutina 3000 graba el fichero:

```
3000 CLS
3010 INPUT"Cual es el nuevo nombre del fichero
        ",fichero$
3020 OPENOUT fichero$
3030 contador=1
3040 WHILE contador<cuenta
3050 WRITE #9,nombre$(contador),telefono$(contad
     or)
3060 contador=contador+1
3070 WEND
3080 CLOSEOUT
3090 PRINT
3100 PEN 3
3110 PRINT"El fichero ha quedado grabado."
3120 INPUT"Pulse ENTER para volver al menu.",ent
     er
3130 RETURN
```

La subrutina 4000 busca en la lista el nombre captado por el teclado:

```
4000 CLS
4010 INPUT"De quien quiere el telefono
        ";amigo$
4020 contador=1
4030 encontrado=0
4040 WHILE contador<cuenta AND encontrado=0
4050 IF amigo$=nombre$(contador) THEN encontrado
     =1:PRINT"El telefono de ";amigo$;" es ";tel
     efono$(contador)
4060 contador=contador+1
4070 WEND
4080 IF encontrado=0 THEN PRINT amigo$;" es desc
     onocido para mi."
4090 PRINT
4100 INPUT"Pulse ENTER para volver al menu.",ent
     er
4110 RETURN
```

El ordenador busca hasta que encuentra el nombre en la lista o hasta que la ha recorrido entera sin éxito. La línea 4080 es útil porque informa al usuario de lo que ha ocurrido; despúes de todo, no es tan difícil equivocarse al escribir un nombre.

La subrutina 5000 permite modificar números de teléfono; utiliza un método de búsqueda similar al anterior:

```
5000 CLS
5010 INPUT"Nombre cuyo telefono quiere cambiar:
        ",amigo$
5020 contador=1
5030 encontrado=0
5040 WHILE contador<cuenta AND encontrado=0
5050 IF amigo$=nombre$(contador) THEN encontrado
     =1:INPUT"Cual es el nuevo numero";telefono$
     (contador)
5060 contador=contador+1
5070 WEND
5080 IF encontrado=0 THEN PRINT amigo$;" es desc
     onocido para mi."
5090 PRINT
5100 INPUT"Pulse ENTER para volver al menu.",ent
     er
5110 RETURN
```

Con esto queda completo el esqueleto del programa; usted puede añadirle ahora los refinamientos que guste.

Ejercicios

1. Este último programa sólo funciona con ficheros ya existentes. Añádale una subrutina para que pueda crear ficheros nuevos. Aplique UPPER$ o LOWER$ a las cadenas que grabe en el fichero para que luego no haya problemas a la hora de buscar nombres en las listas.
2. Añada una rutina que permita al usuario repasar rápidamente el fichero; debe aparecer en la pantalla un registro cada vez que se pulse una tecla.
3. Incluya una rutina que permita añadir nuevos registros o suprimir otros.
4. Reescriba el programa para adaptarlo a otro tipo de datos.

CONCLUSIÓN

Esperamos que este libro haya aclarado algunas de sus dudas sobre el ordenador Amstrad. No es posible llegar a saberlo todo acerca de un ordenador, como demuestra la proliferación de revistas de informática que se observa en los quioscos. Todos los meses alguien descubre una solución nueva para un antiguo problema, o una capacidad insospechada en el ordenador.

En el limitado espacio de que disponíamos, hemos tratado de presentar y explicar el mayor número posible de instrucciones del BASIC del Amstrad. Hay omisiones, inevitablemente. En algunos casos podríamos habernos extendido mucho más; por ejemplo, de las instrucciones gráficas sólo hemos explicado lo más elemental. Pero, por otra parte, tampoco se puede hablar de todo en un libro de introducción como es este.

Si usted quiere profundizar en el conocimiento de su ordenador, por ahora no hay más información oficial que el Manual del Usuario (aparte del Manual del Firmware, que no habla de BASIC, sino del sistema operativo). Hay otros libros de nivel más o menos equivalente al de éste y muy pronto se publicará una revista dedicada exclusivamente a este ordenador.

Para terminar, recuerde que todos los autores de libros y artículos de revistas han sido alguna vez principiantes como usted. Lo que ahora le parece tan complicado muy pronto le resultará increiblemente fácil. ¡Disfrute practicando!

Índice

PROGRAMACION BASIC CON AMSTRAD

El ordenador AMSTRAD es una máquina muy potente, con funciones gráficas y sonoras muy avanzadas. El manual del usuario no puede dar más que un repaso superficial a las características del ordenador; en cambio, en este libro nos tomamos las cosas con más calma, para yudar tanto a los principiantes como a los usuarios más expertos.

Los dos primeros capítulos van destinados al principiante absoluto. El resto del libro presenta la mayor parte de las instrucciones de BASIC disponibles en el AMSTRAD. En los últimos capítulos se abordan temas más especializados, tales como la utilización del sonido y manejo de fichero de datos. En todos los capítulos se incluyen abundantes ejemplos y ejercicios.

El autor

Wynford James se dedica en la actualidad para escribir material educativo (incluidos programas) para una importante empresa de microordenadores. Antes ha trabajado como autor técnico para ICL. También ha sido profesor de matemáticas y ha participado en el desarrollo de estudios informáticos.



Avda. del Mediterráneo, 9 28007 MADRID

PROGRAMACION BASIC CON AMSTRAD

Wynford James


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/